IL COMPUTER È FACILE PROGRAMMIAMOLO INSIEME

GIANNI BECATTINI

GIANNI BECATTINI

IL COMPUTER È FACILE PROGRAMMIAMOLO INSIEME

a mio Padre

edizioni CD - Bologna - via Boldrini, 22
supplemento a **CQ ELETTRONICA**

I LIBRI DELL'ELETTRONICA

I - E. ACCENTI - *Dal transistor ai circuiti integrati* - 1969

II - A. BARONE - *Il manuale delle antenne* - 1971

III - L. RIVOLA - *Strumenti di misura e unità di alimentazione* - 1972

IV - L. RIVOLA - *Trasmettitori e ricetrasmettitori* - 1972

V - M. MICELI - *Come si diventa CB e Radioamatore* - 1976

VI - U. BIANCHI - *Radiosurplus ieri e oggi* - 1982

VII - G. BECATTINI - *Il computer è facile programmiamolo insieme* - 1983

La composizione tipografica di questo volume è stata effettuata direttamente dall'Autore con un Personal Computer e **volutamente** mantenuta dall'Editore, con il solo tramite della riduzione fotografica.

TIPOGRAFIA-LITOGRAFIA LAME - BOLOGNA

CAPITOLO I
Il computer è facile

Questo e' un testo semplice dedicato a tutti coloro che pur non disponendo di preparazione specifica desiderano avvicinarsi al nuovo affascinante mondo degli Home e Personal computers. Per la sua lettura non e' indispensabile alcun requisito, se non quello dell'interesse per la materia. Con in mente questi obiettivi, ho cercato per il possibile di rendere la lettura piacevole ed interessante.

Se questo libro sapra' suscitare nel lettore il desiderio di approfondire gli argomenti trattati e di accrescere le proprie conoscenze nel campo della informatica, potro' dire che ha raggiunto perfettamente il suo scopo.

I computer sono ormai ovunque. Smitizzati dal loro ruolo di "mostri sacri", sono diventati i piu' validi strumenti di lavoro per ogni tipo di attivita', da quella del piu' modesto commerciante (fig.1) a quella delle piu' grandi aziende.

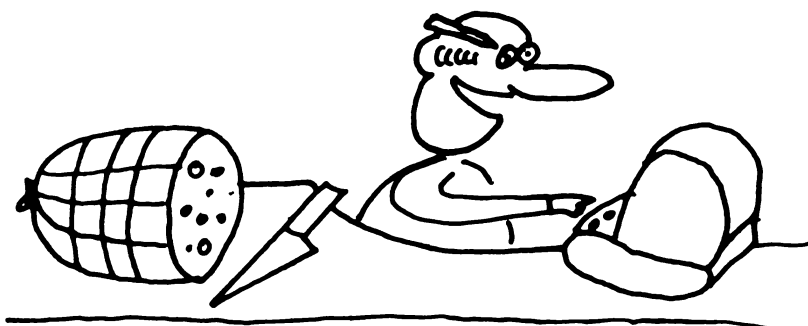



figura 1

C'e' chi dice che in un futuro molto prossimo non saper utilizzare un elaboratore elettronico sara' ritenuto pari a cio' che oggi rappresenta il non saper usare il telefono.

Vi devo subito parlare di programmazione perche' il computer e' una "macchina" un po' diversa dalle altre che gia' conosciamo (per esempio il telefono, il televisore, la macchina per scrivere, il giradischi, ecc.); queste ultime infatti sono costruite di ferro, transistori, plastica, manopole, come un computer, ma, per funzionare, gli occorrono solo una corrente elettrica e l'azionamento di alcuni comandi (tasti, levette, ecc.). In definitiva, sono interamente costituite da **hardware** (parola americana che significa "ferramenta", "ferraglia", cioe' roba che si vede e che si tocca). Il computer invece e' fatto si' di hardware, ma non basta dargli corrente e premere tasti o levette per farlo funzionare, ma bisogna dirgli "cosa fare"; per dare un'anima a quella "ferraglia" occorre il **software**, cioe' il complesso di istruzioni, di programmi, appunto, che al computer stesso danno vita e fanno si' che il suo hardware svolga le funzioni che noi gli abbiamo comunicato tramite il software. Programmi dunque. Per migliore precisazione, classifichiamoli:

- sistemi operativi
- linguaggi
- programmi applicativi

I **sistemi operativi** li allestisce il Costruttore e rendono l'hardware capace di compiere le funzioni di base ("parlare" con le sue unita' costituenti e gestirle: tastiera, dischi, nastri, stampanti, ecc.). I **linguaggi** servono a rendere possibile e agevole il colloquio dell'uomo con la macchina; ci soffermeremo ancora a lungo su questo un po' piu' avanti. Il BASIC e' un linguaggio. Anche i linguaggi sono di norma allestiti da un Costruttore. Una volta che il computer sia dotato di sistema operativo (quindi capace di "muovere" testa, gambe, occhi e braccia) e provvisto di un linguaggio (quindi capace di capire cosa gli dico e di dire lui a me cosa sta facendo in modo a me comprensibile), si possono realizzare i **programmi applica-**

tivi. Questi consentono al computer di effettuare una specifica attivita', quindi di avere una applicazione di pratica utilita', ad esempio di tenere la contabilita' del mio conto corrente personale o della mia azienda. I programmi applicativi possono essere "packages", ossia pacchetti gia' preparati dal Costruttore, che adottero' se fanno al caso mio, oppure "programmi utente" che io stesso mi sono costruito usando uno dei linguaggi che il mio computer capisce. Qui impareremo a farci i nostri programmi usando il linguaggio BASIC che e' uno dei piu' universali e di cui sono dotati tutti gli home e personal computers.

Sono ancora molti quelli che ritengono la programmazione un'arte difficile riservata agli iniziati. Tre errori: la programmazione non e' un arte, non richiede infatti doti innate ma puo' essere appresa da tutti; la programmazione non e' difficile ma costituisce una attivita' piacevole ed intelligente; terzo, non e' riservata agli iniziati, anzi, per divenire programmatori non e'richiesta alcuna conoscenza preliminare, ne' di matematica, ne' di elettronica; basta il buon senso e la buona volonta'.

Imparare divertendosi

Imparare e' spesso sinonimo di fatica; imparare la programmazione invece e' cosi' appassionante che non solo la fatica non si sente ma le ore volano via rapide e sembrano minuti. Nel corso di questo volume, che vuole essere fondamentalmente un testo semplice, supporremo nulle tutte le vostre conoscenze. Siamo certi che fino dalle prime pagine troverete spunti sufficienti ad indurvi a proseguire nella lettura ed a desiderare di approfondire i concetti illustrati.

Home computers

I motivi principali della enorme diffusione degli elaboratori sono due: la riduzione dei costi, dovuta all'evoluzione della tecnologia, e la sempre maggiore facilita' con cui essi possono essere programmati. E' quindi nato il fenomeno, impensabile solo pochi anni fa, dell' "home computing", cioe' dell'elaborazione dati

casalinga, del computer personale che costa ormai anche meno di una buona macchina fotografica. Questa evoluzione non e' limitata all'ambito familiare ma si e' estesa e si estende in tutte le direzioni, tanto che non e' ormai raro il caso di scuole elementari ove si insegnano i rudimenti della programmazione.

Le applicazioni dell'home computer sono infinite. Abbiamo conosciuto persone che lo usano per gestire il proprio bilancio personale, verificando il proprio conto corrente e controllando finalmente i complessi tabulati della banca(fig.2), o per pianificare le proprie spese. Altri lo usano per giocare a scacchi (fig.3), e si trovano davanti ad un forte avversario, a "master mind" o ad altre infinite varieta' di giochi intelligenti mentre quelli dal temperamento piu' artistico lo impiegano per comporre e suonare melodie (fig.5).

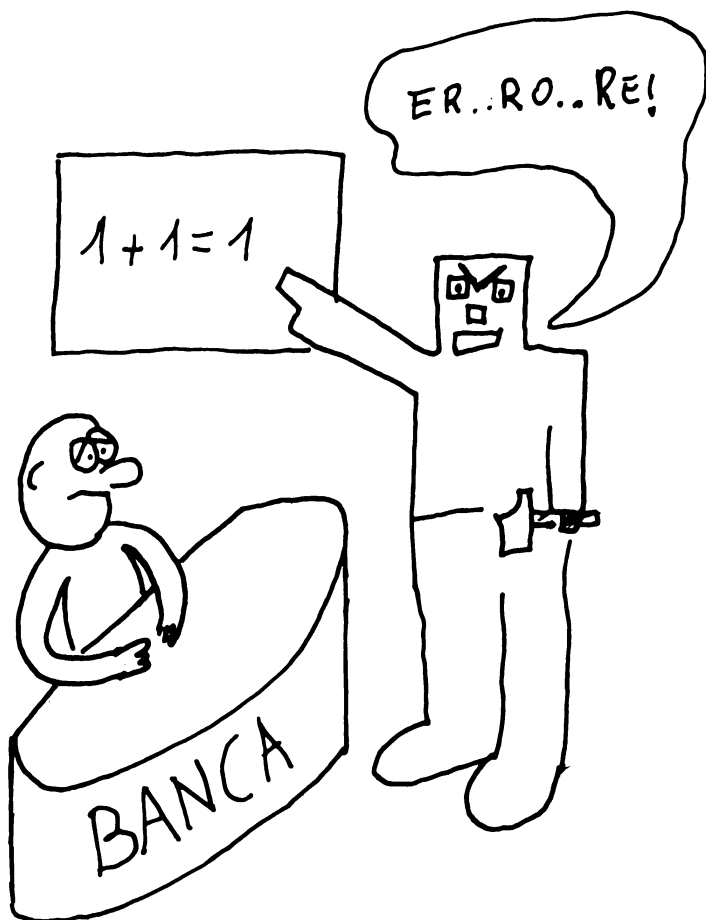



figura 2

Con opportuni programmi, l'home computer puo' diventare il piu' paziente maestro, facilitando l'apprendimento delle nozioni piu' ostiche ai vostri figli; favorendone la familiarita', cancellera' per sempre dalle loro menti l'idea che il computer sia un qualcosa di estraneo e pericoloso (fig.6).

Le risorse della mente

L'apprendimento della programmazione si trasforma in una estimabile ricchezza a livello personale. L'evoluzione del mondo del lavoro fa si' infatti che, come abbiamo gia' accennato all'inizio, l'informatica si sia estesa anche in ambienti dai quali era tradizionalmente esclusa: non e' piu' infatti una rarita' il vedere sul bancone del pizzicagnolo, accanto alla affettatrice di salumi, un piccolo elaboratore che gestisce la contabilita' del negozio, le scorte del magazzino e magari un'insegna pubblicitaria ottenuta con un televisore a colori. Anche le grosse aziende, gia' dotate di reti informative sofisticate e costose, hanno ormai appreso la validita' del piccolo computer, da affiancare al lavoro dei loro dipendenti per renderlo meno tedioso e piu' produttivo.

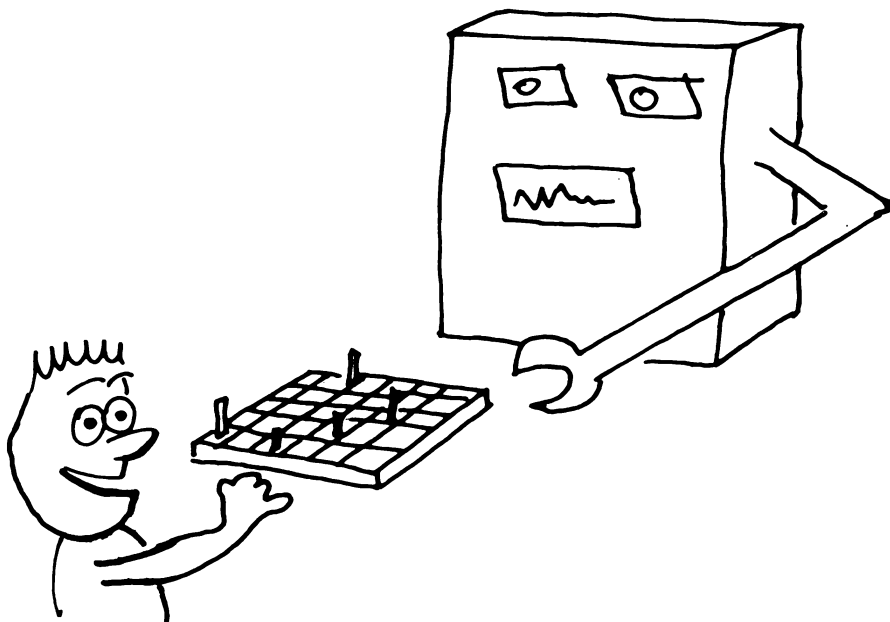



figura 3

A differenza di altre materie, la programmazione puo' essere appresa da chiunque, senza corsi di formazione preliminari ne' una cultura superiore. E' cioe' la materia dove ciascuno puo' diventare autodidatta. E la conoscenza della programmazione contribuisce in maniera determinante all'accrescimento della propria statura professionale, in qualsiasi attivita' (fig.10).

Cosa e' un computer

Ridotto ai minimi termini, un computer e' una macchina elettronica che (fig.7):

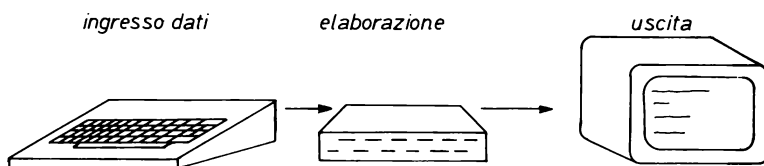



figura 4

1) Riceve dati, che puo' anche memorizzare, attraverso una tastiera.

2) Li elabora.

3) Mostra i risultati su un video.

Esempi: in un **gioco**:

1) Riceve comandi dalla tastiera.

2) Li elabora.

3) Mostra i risultati sul video (ad esempio ci indica se, con un certo alzo, siamo riusciti a colpire un bersaglio).

Altro esempio, una contabilita':

1) Riceve dati, attraverso la tastiera, sulle fatture in arrivo e in partenza e sui pagamenti.

2) Li elabora.

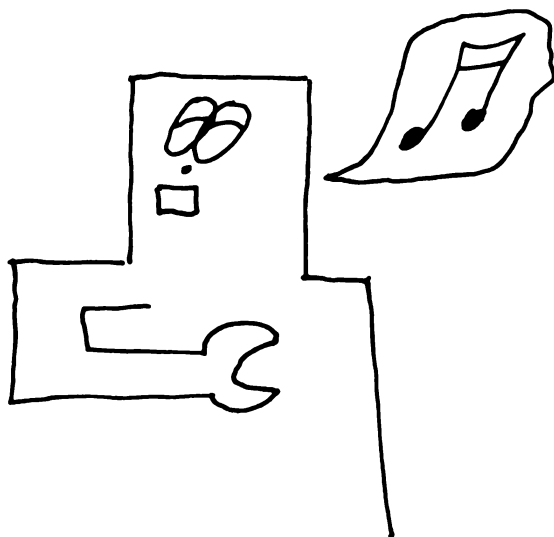



figura 5

3) Fornisce in uscita una varieta' di risposte (quanto abbiamo venduto questo mese? quanto ci deve pagare il cliente Chiodis? Quanto dovremo pagare di IVA al prossimo mese?)

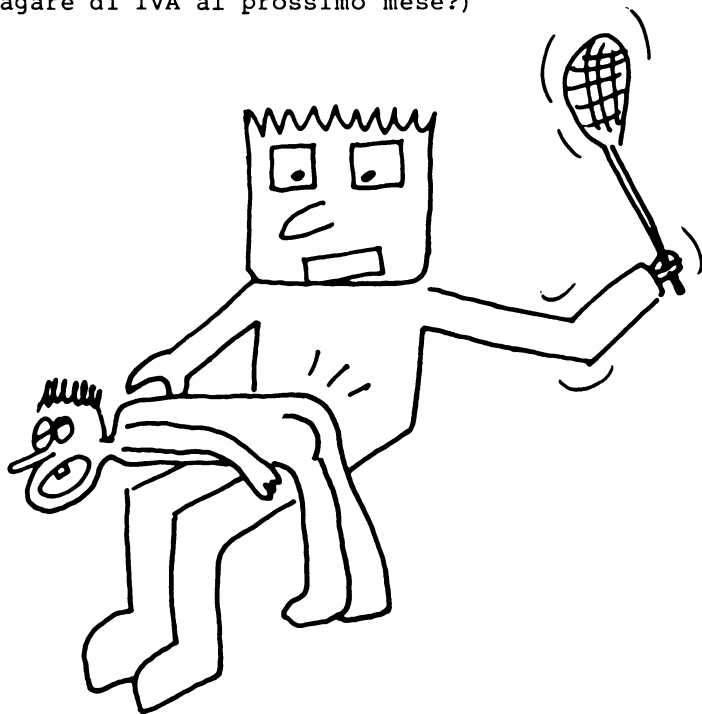



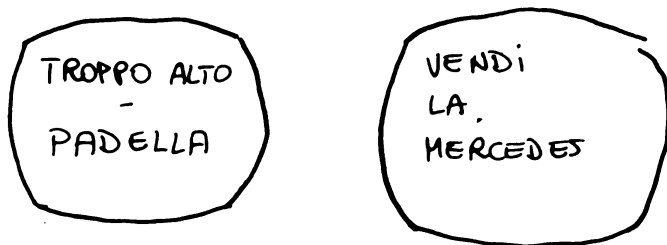
figura 6

Per quanto ci si possa scervellare tutte le applicazioni si riducono alle fasi ora viste **"Introdu-**

zione-Elaborazione-Uscita risultati" (o combinazione di esse).

Altri esempi

Non e' detto che il ciclo Introduzione-Elaborazione debba necessariamente concludersi dopo l'Uscita risultati; puo' riprendere con una nuova introduzione, seguita ancora da una elaborazione e da una uscita risultati. Nell'esempio del gioco potremmo avere delle risposte dalla prima elaborazione tali da influenzare le successive introduzioni. Se il gioco consiste nello sparare con un cannone ad un bersaglio, da un tiro sbagliato potremo tentare con una nuova introduzione del valore dell'alzo. Se la previsione di bilancio indica un crack possiamo tentare di ridurre le spese (fig. 7). Ma lo schema e' sempre il solito "Introduzione (tramite la tastiera) - Elaborazione - Uscita (su video)".

figura 7

Elaboratori reali

Una grossa differenza tra la schematizzazione ora enunciata e gli elaboratori reali e' data (fig.8) dalla possibilita' di questi ultimi di accettare ingressi da apparecchi diversi dalla tastiera (ad esempio, da strumenti scientifici, da linee telefoniche ecc.) e di fornire le risposte su uscite diverse dal video (ad esempio verso una macchina che scriva le risposte su carta, detta stampante, verso una linea telefonica, verso un tabellone luminoso, o addirittura verso macchine operatrici, come un tornio od una fresa). Gli home computer, dei quali principalmente ci occuperemo, possiedono molti tipi di unita' di ingresso e di uscita (dette interfacce) appositamente studiate in vista della loro applicazione.

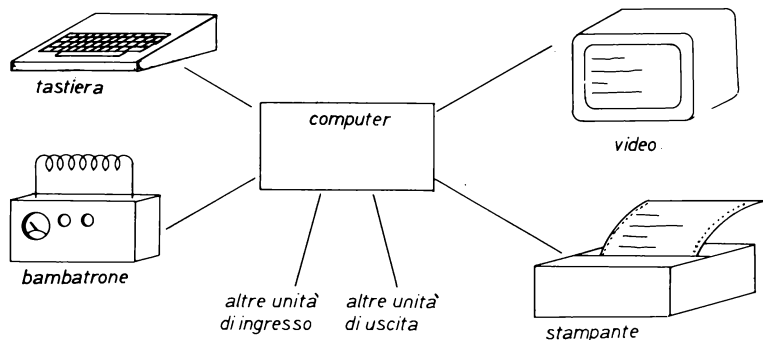



figura 8

La figura 11 mostra, come esempio, un elaboratore reale per applicazioni professionali di elevate prestazioni. Si tratta del Sistema 4 della General Processor, interamente progettato e costruito in Italia.

L'elaborazione

E' quindi chiaro che, se si escludono le parti di ingresso e di uscita, resta nel computer la sola parte che esegue le elaborazioni; questa parte, che in un certo senso costituisce il computer vero e proprio, e' detta Unita' Centrale di Elaborazione (siglata CPU, dall'inglese Central Processing Unit). L'elaborazione non avviene in modo fisso; siamo noi che dobbiamo indicare al computer la procedura per compierla.

Macchine stupide

Il computer e' infatti una macchina stupida (fig. 9). Da solo non e' in grado di fare un bel niente. E' solo un banale, velocissimo e perfettissimo esecutore ottuso di ordini. Qualcuno potrebbe obiettare: "Eppure io ho visto un computer che dava risposte sensate, che faceva questo, che faceva quello....come puoi affermare che e' stupido?". E' semplice. Tu non hai visto un computer intelligente. Hai solo visto un computer che elaborava in modo intelligente delle informazioni che gli venivano fornite. L'intelligenza che tu

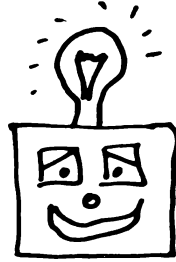



figura 9

hai creduto di vedere e' solo quella della persona che ha dato al computer le istruzioni sul modo in cui eseguire l'elaborazione. In altre parole hai visto l'intelligenza di chi lo ha programmato. Potete subito scolpire nel muro di casa vostra la

Regola aurea dell'informatica

"Nessun computer e' piu' intelligente
di chi lo ha programmato"

La programmazione

La programmazione e' quindi l'operazione di fornire al computer le istruzioni sul come

figura 10

comportarsi in funzione dei dati che gli vengono introdotti. Ad esempio, un computer puo' essere programmato perche':

1) Si inventi un numero a caso senza pero' mostrarlo (fase di elaborazione).

2) Proponga attraverso il video la domanda "Indovina che numero ho pensato?" (fase di uscita)

3) Accetti la risposta attraverso la tastiera (fase di ingresso)

4) Controlli la risposta (fase di elaborazione) e scriva "Bravo!" se la risposta e' esatta (fase di uscita) o "Tenta ancora!" in caso contrario (fase di uscita).

Attraverso questo libro vedrete l'esempio sopra riportato tradotto in pratica, ed altri ancora piu' complessi. Vedrete inoltre quanto sia facile risolvere con un home computer problemi apparentemente molto difficili.

I linguaggi

Il computer purtroppo e' una macchina molto rudimentale. Le istruzioni che di per se'

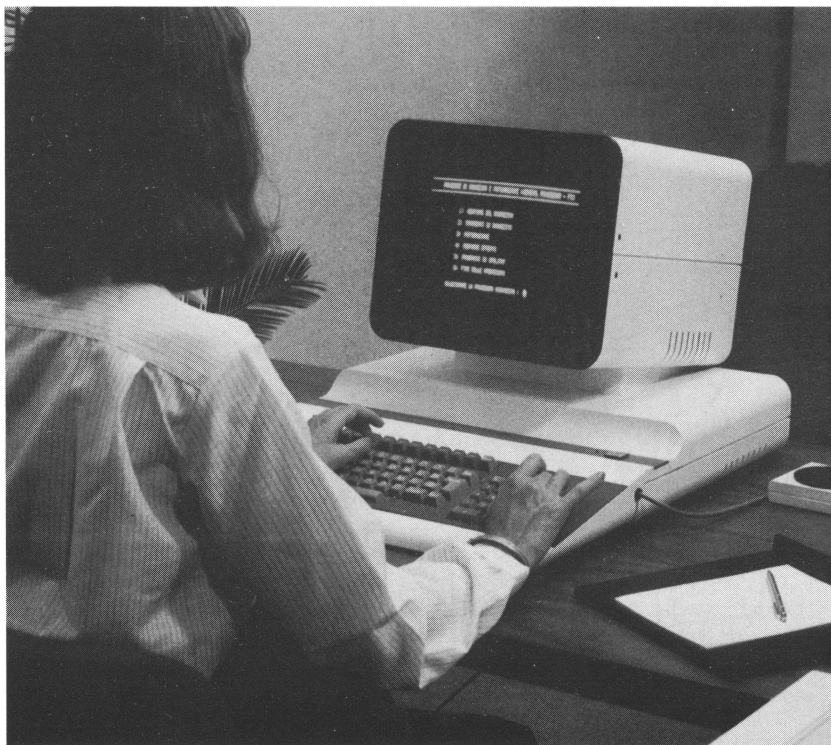



figura 11
Un moderno elaboratore italiano: si tratta del GPS-4 della General Processor (per gentile concessione della General Processor).

puo' eseguire sono quasi banali; questo fa si' che una operazione anche semplicissima richieda non una ma centinaia e centinaia di istruzioni elementari. Il nostro desiderio sarebbe invece quello di possedere degli elaboratori che accettassero direttamente il nostro linguaggio; ad esempio, per calcolare l'area di un rettangolo, sarebbe bello potergli semplicemente dire

DIMMI UN PO' QUANTO FA 334 x 566.

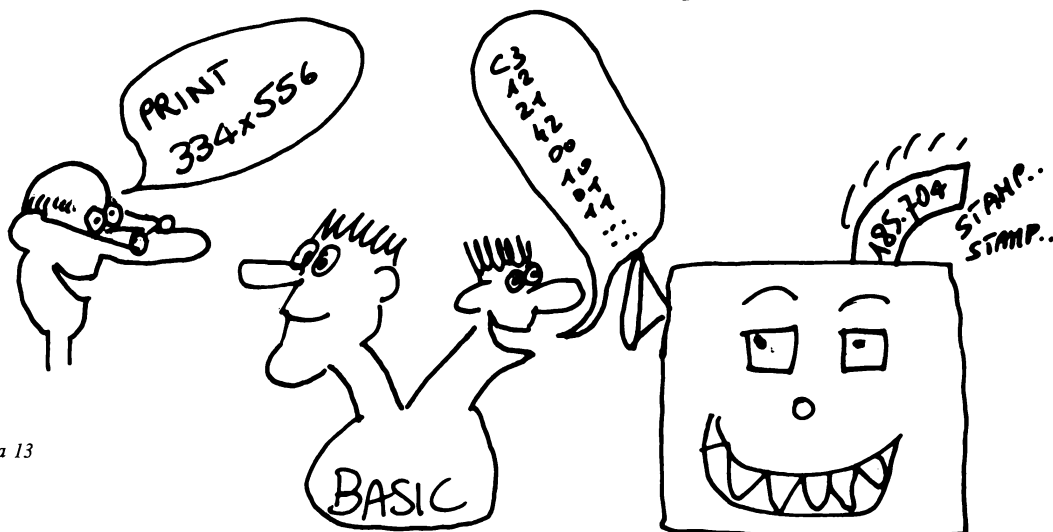
Purtroppo questa frase e' incomprensibile dal computer (fig.12). Per ovviare a questa "incompatibilita'" tra l'operatore uomo, che vorrebbe un

figura 12

linguaggio molto simile al parlato, e la macchina, che desidererebbe invece solo complesse file di istruzioni elementari composte di 0 ed 1, sono stati inventati i **linguaggi di programmazione** (fig.13), tra i quali si trova il BASIC, che studieremo assieme nel corso del presente volume.

Come e' nato il BASIC

L'origine dei linguaggi di programmazione e del BASIC in particolare e' dovuta a motivi commerciali. Quando infatti i primi elaboratori co-

figura 13

minciarono a diffondersi (in senso molto relativo, rispetto al boom dei giorni nostri), coloro che li vendevano o che vendevano i loro servizi diffusi tramite reti telefoniche si trovarono di fronte ad un grave problema: "come vendere delle macchine cosi' difficili da programmare? (anche se potenti ed innovatrici)" Finalmente qualcuno ebbe un'idea brillante. "Inventiamo un linguaggio piu' simile possibile a quello umano e facciamo si' che sia lo stesso elaboratore, tanto bravo nei lavori monotoni e ripetitivi, ad effettuare la traduzione da questo linguaggio nel <u>suo</u> linguaggio". Nacque cosi' il BASIC ed inizio' una nuova era, un era in cui tutti potevano imparare ad utilizzare le immense risorse di un grande elaboratore. Con l'avvento dell'home computer si sono poi riscoperte le immense potenzialita' del BASIC, cosi' semplice da usare, ed esso e' cosi' divenuto il linguaggio per eccellenza per tutti coloro che necessitano

dell'informatica pur non costituendo essa la loro principale attivita' o cultura. Alla fine della lettura di questo libro, vi sarete resi conto della facilita' di apprendimento del BASIC; con esso e' possibile risolvere problemi di ogni tipo, gestionali, scientifici, di hobby ecc.

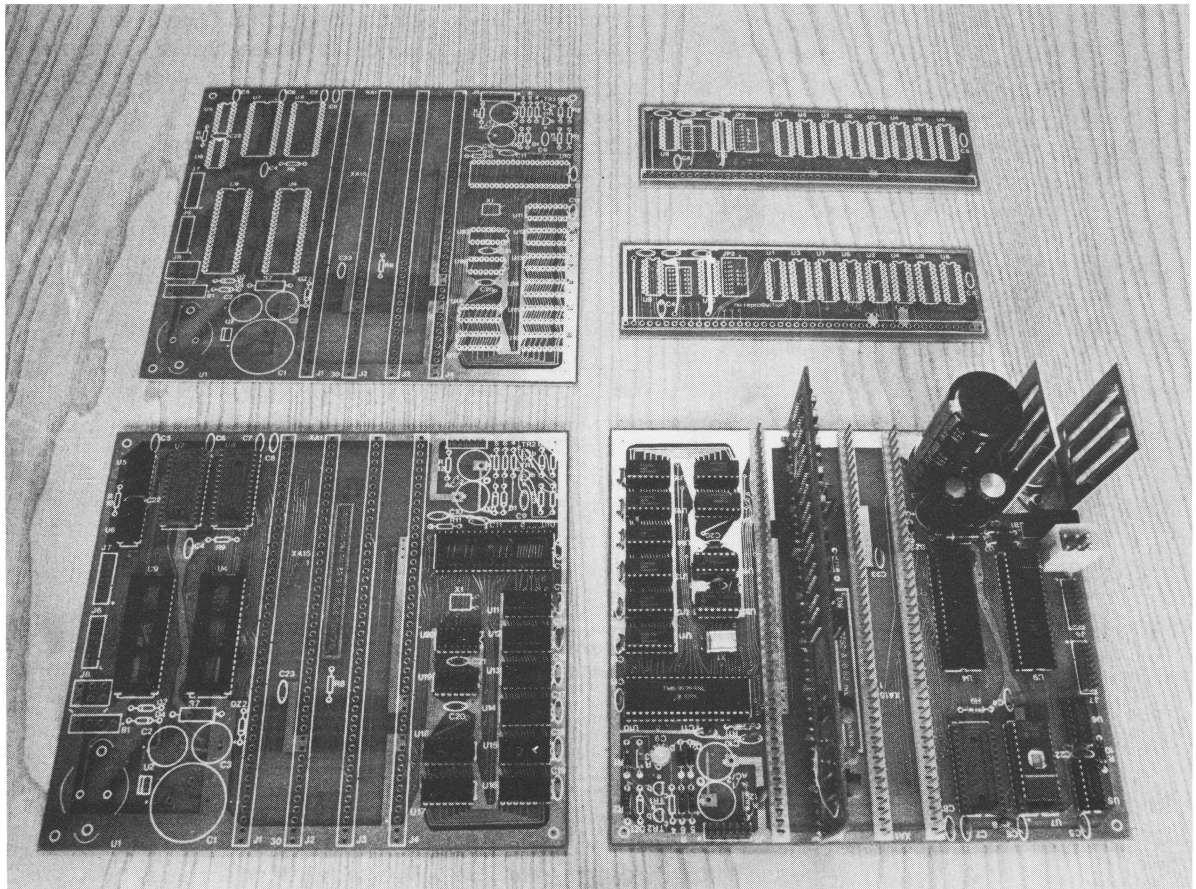



figura 14
Il microcomputer G5 in tre diversi stadi: in alto i soli "circuiti stampati", a sinistra in basso all'inizio del montaggio e a destra praticamente finito.
Costruire personalmente il proprio elaboratore può essere una "avventura" estremamente appassionante.
Tutte le informazioni necessarie alla realizzazione sono comparse sulla rivista **CQ ELETTRONICA** *nei numeri 1/83, 2/83 e 4/83.*

Nota Il BASIC odierno e' molto piu' completo di quello originariamente ideato dai suoi creatori.

Questo perche' nel passare del tempo, molti altri progettisti hanno ideato modifiche ed aggiunte che lo hanno portato alla fase attuale. In seguito a cio' esistono ormai diverse versioni di BASIC, fondamentalmente simili ma con differenze minori. In questo manuale ci riferiremo sempre al BASIC, realizzato dallo stesso autore ed utilizzato nel G5, il piccolo ma potente home computer presentato sulle pagine della rivista **cq elettronica** e che tutti possono autocostruire con poca spesa e certezza di risultato. Le istruzioni di macchine diverse possono differire anche se la trattazione base rimane ugualmente valida. Passare ad un altro elaboratore e' simile al passare da una vettura all'altra: la tecnica di guida e le regole da rispettare sono le stesse anche se l'interruttore delle luci, il freno a mano e l'accendisigari si trovano in posizione differente o con indicazione in lingue diverse.

CAPITOLO II
Più facile di una calcolatrice

Cominciamo dal basso

Supponiamo di sedere davanti al nostro computer. Per attivarlo basta accenderlo e premere il tasto RESET. Sul video comparira' una scritta che indica per solito la marca del microcomputer in uso e la specifica versione del BASIC. Potra' anche apparire una domanda che chiede se il programma e' nuovo (N) o vecchio (V).

Per adesso rispondiamo N (programma nuovo) in quanto non abbiamo ancora programmi registrati in memoria. La risposta V (programma vecchio) andra' fornita nel caso in cui si desideri ritrovare nella memoria del microcomputer un programma precedentemente memorizzato. Qualunque cosa la macchina stia facendo, premendo RESET si ottiene sempre l'effetto di ripulire il video e di far comparire la domanda ora vista.

Avendo conosciuto le nostre intenzioni il G5 mostrera' sul video il segno

:

questi "due punti" indicano semplicemente che la macchina e' pronta ed e' in attesa di ordini.

Facile come una calcolatrice

Il nostro computer puo' essere usato, grazie al BASIC, come una normale calcolatrice. Vogliamo provare? bene, vediamo allora come ottenere la somma di 12345 e di 54321. Basta scrivere

```
PRINT 12345+54321
```

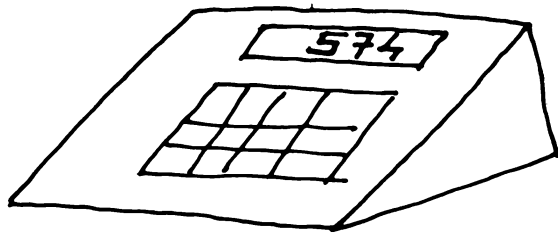



figura 15

e quindi il tasto RETURN (quello grande a destra) perche' la macchina scriva immediatamente il risultato sullo schermo. Analizziamo cio' che abbiamo fatto:

PRINT

Primo: abbiamo dato il comando PRINT che in inglese vuol dire STAMPA.

Secondo: abbiamo indicato cosa il computer deve stampare, cioe' 12345 piu' 54321.

Terzo: col tasto RETURN abbiamo indicato alla macchina di aver finito di battere il rigo con il comando e di iniziarne l'esecuzione. In risposta la macchina ha analizzato ed eseguito il nostro comando ed ha "stampato" il risultato sul video. Ad essere precisi sia la parola "stampato" che lo stesso comando "PRINT", cioe' "STAMPA", non sono proprio azzeccati; l'operazione di "stampare" infatti si riferisce piu' al trasferire delle scritte sulla carta che non su un video. Questa imprecisione ha delle ragioni storiche; infatti i primi terminali dei vecchi computer non erano del tipo video bensi' delle telescriventi con rotolo di carta. Con l'evoluzione della tecnica e la maggiore diffusione dei video, la parola, un tempo piu' appropriata, e' stata conservata e la si usa ormai correntemente per quanto inesatta.

Il magico tasto RETURN

E' importante notare che il computer ignora tutto quanto scriviamo fino alla pressione del tasto RETURN. Questo, che trae il suo nome dal tasto di RITORNO A CAPO delle macchine per scri-

vere, equivale quindi a dire alla macchina: "Ho

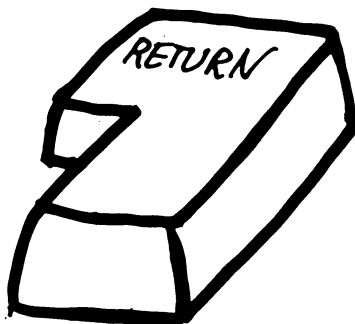



figura 16

scritto un messaggio per te; leggilo ed eseguilo". **E' sempre indispensabile quindi premere il RETURN alla fine di ogni rigo;** diversamente il computer non eseguira' i nostri comandi ma restera' in attesa di nuove battute. Non ripeteremo piu' quindi che e' necessario premerlo e sara' sempre sottinteso.

E se sbaglio?

Finche' non si e' premuto il RETURN e' sempre possibile effettuare delle correzioni di cio' che si e' battuto. Immaginiamo di avere digitato PRONT invece di PRINT. Niente paura; per cancellare basta premere il tasto "[-" (freccia a sinistra); ogni volta che lo si preme la posizione di scrittura (segnalata da un trattino luminoso sul video) "arretra" di un posto e cancella i caratteri eventualmente presenti. Basta quindi tornare indietro fino a cancellare i caratteri errati e ribattere come niente fosse successo.

La macchina segnala gl errori

Attenzione pero': le correzioni sono possibili solo prima di avere premuto il RETURN. Una volta infatti che sia stato schiacciato, la macchina riconoscera' i nostri errori e, se possibile, ce li segnalera'. Perche' se possibile? Perche', ad esempio, l'elaboratore non trova difficolta' a riconoscere la parola PRONT come errata; infatti non esiste in BASIC un comando PRONT.

Ma non puo' certo accorgersi se invece di scrivere

PRINT 12345+54321

abbiamo scritto

PRINT 12348+54321

in quanto nessuna macchina puo' essere a conoscenza delle nostre effettive intenzioni!.

Il primo errore, quello di avere scritto PRONT, viene segnalato con la frase

ERRORE DI LINGUAGGIO.

Altri messaggi sono previsti per altri tipi di errori come vedremo meglio in futuro.

Esempi

Supponiamo adesso di essere stati al grande magazzino e di aver acquistato 12 bicchieri da 1200 lire, 6 scatole di chiodi da 560 lire e 6 lattine di birra da 600 lire. Il totale della spesa puo' essere calcolato cosi'

totale = 12 x 1200 + 6 x 560 + 6 x 600

figura 17

In BASIC non ci si deve scervellare per determinare l'ordine in cui devono essere battuti i vari tasti come avviene sulle calcolatrici. Basta battere

PRINT 12*1200+6*560+6*600

(al solito seguito da RETURN, non lo ripeteremo piu'!) e subito compare sul video il risultato. Unica osservazione: in BASIC si usa l'asterisco (*) invece del x di moltiplicazione e la barra (/) al posto del diviso (:).

Altro esempio:

Calcolare quanto olio contiene, in peso, una vasca rettangolare avente le seguenti misure

lunghezza = m 2.45
larghezza = m 1.38
profondita' = m 0.85

supponendo che il peso specifico dell'olio sia kg 0.850 per ogni decimetro cubo.

Ricordiamo che:

1) Il volume della vasca e' dato dal prodotto dei tre lati.

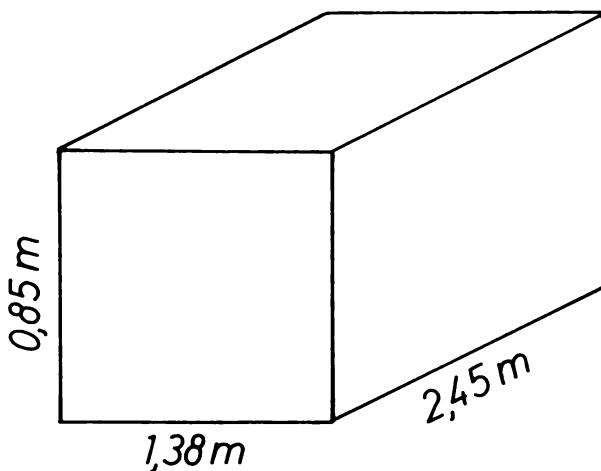



figura 18

2) In un metro cubo ci stanno 1000 decimetri cubi.

La risposta e' data quindi da

2.45 x 1.38 x 0.85 x 1000 x 0.850

ed il risultato puo' essere ottenuto semplicemente con la frase

```
PRINT 2.45*1.38*0.85 * 1000 * 0.850
```

la risposta e' 2442.77 Kg. Si osservi che e' indifferente inserire degli spazi tra i numeri ed i segni di operazione.

Espressioni aritmetiche

La parte che segue la parola PRINT, cioe' quella che indica l'operazione vera e propria, e' detta <u>espressione</u> <u>aritmetica</u>. L'espressione aritmetica puo' essere composta, al limite, da un solo numero (caso abbastanza inutile nella frase PRINT):

```
PRINT 5
```

ma puo' diventare, se necessario anche molto complessa. In ogni caso valgono le regole della aritmetica: prima si eseguono le moltiplicazioni e le divisioni, per ultime le somme e le sottrazioni. Quindi

```
PRINT 3+8*4
```

da' come risultato 35 (4x8=32 piu' 3 = 35) e non 44. L'ordine delle operazione puo' tuttavia essere alterato con l'uso delle parentesi;

```
PRINT (3+8)*4
```

da' risultato 44. Quindi, quando ci sono delle parentesi, che possono essere anche innestate una dentro l'altra, si iniziano i calcoli dalle parentesi piu' interne.

Calcolatrice parlante

Abbiamo fin qui esaminato l'uso del BASIC a mo' di calcolatrice. Abbiamo gia' visto un grosso vantaggio almeno nei confronti della maggior parte delle cacolatrici: che le espressioni aritmetiche possono essere battute esattamente come le scriviamo sulla carta, senza preoccuparsi di complesse sequenze di digitazione o di tasti speciali. Vediamone subito un altro con un esempio.

Sia da calcolare la superficie di un cerchio avente un raggio di 45.8 cm. La formula per calcolarla la si impara in terza elementare ed e'

raggio al quadrato x 3.14

dove 3.14 e' il numero fisso detto pi greco. In BASIC l'elevazione al quadrato la si ottiene moltiplicando un numero per se' stesso. Quindi

```
PRINT 45.8 * 45.8 * 3.14
```

la risposta e' 6586.59 cmq.

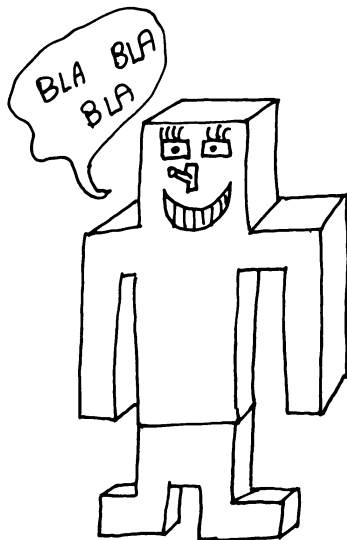



figura 19
Calcolatrice parlante.

Il BASIC ci consente di fare di piu'; ci consente di indicare esattamente cosa ha fatto, con un grande vantaggio per la chiarezza nella risposta. Ci permette infatti di scrivere

```
PRINT "L'area cercata vale cmq ";45.8*45.8*3.14
```

e di avere come risposta

L'area cercata vale cmq 6586.59

Il nostro home computer sta gia' diventando un super calcolatore tascabile; e' infatti in grado di dirci cosa gli abbiamo fatto calcolare. Una osservazione ovvia: il computer (e qui si torna al discorso della macchina stupida), non sa assolutamente cosa noi scriviamo dopo il comando PRINT tra le virgolette; si limita a recepirlo ed a trascriverlo prima di mostrare il risultato. Se noi avessimo scritto

PRINT "La gallina coccode'"; 45.8*45.8*3.14

avremmo avuto in risposta

La gallina coccode' 6586.59.

A che serve

Gia' ci sara' qualcuno che osserva: "a che pro fargli scrivere la risposta preceduta da una frase che io stesso ho battuto un istante prima? Ci mancherebbe che non mi ricordassi cosa sto facendo dopo neanche una frazione di secondo!". Pazienza, amici, pazienza! Tutto cio' sara' ben chiaro in seguito.

Le memorie

Alcune calcolatrici hanno dei tasti speciali che servono per memorizzare dei valori da usare nei calcoli successivi. Anche il BASIC puo' memorizzare dei valori, e con molta maggiore facilita' e flessibilita'. Rifacendosi all'esempio precedente, immaginiamo il caso in cui si debbano calcolare le superfici di non uno ma di molti cerchi. E' possibile evitare di ribattere tutte le volte il valore di pi greco (3.14). Basta eseguire il comando

LET

LET P = 3.14

che significa:

1) crea una casella di memoria e dagli nome P

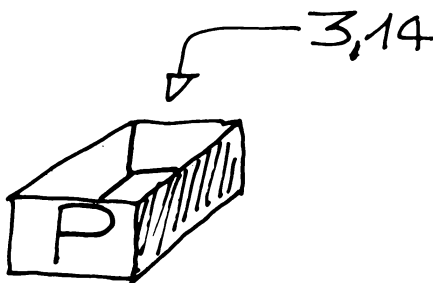



figura 20

2) immagazzina il numero 3.14 nella casella di nome P (LET vuol dire ASSEGNA, quindi "ASSEGNA alla cella P il valore 3.14").

Dopo questo comando, il BASIC ricordera' che al nome P e' associato il numero 3.14. Per verificarlo basta battere

PRINT P

e la risposta sara' 3.14. D'ora in poi il nome P potra' essere sempre usato nei calcoli e potremo quindi scrivere

PRINT 45.8*45.8 * P

esattamente come prima scrivevamo

PRINT 45.8*45.8*3.14.

Il valore di P restera' inalterato finche':

1) non lo si alteri con una nuova frase LET P=...

2) non si impartisca un comando (vedremo poi) che provochi la cancellazione di tutte le celle di memoria.

Il microcomputer G5, cui ci stiamo riferendo, ha infatti la memoria "continua" che permane anche dopo lo spegnimento. In quasi tutti gli altri elaboratori, non provvisti di questa caratteristica, si ha la cancellazione totale alla mancanza della tensione di alimentazione.

Variabili

La P dell'esempio precedente si chiama **variabile.** In BASIC possono essere create molte variabili tenendo presente che i loro nomi possono essere cosi' composti:

1) possono essere di uno o due caratteri

2) il primo carattere deve essere una lettera

3) il secondo carattere puo' essere una lettera od un numero

sono quindi nomi validi di variabili

P
P1
A
B
C7.

Ancora sulla frase LET

Nella frase LET, che ci consente di creare le variabili o di modificarne il valore, consente anche di memorizzare risultati intermedi di calcoli. Ad esempio, riprendiamo il caso della spesa al grande magazzino. Ricordate come l'avevamo risolto?

```
PRINT 1200*12 + 600*6 + 560*6
```

E' la stessa cosa fare

```
LET A=1200*12
LET B=600*6
LET C=560*6
PRINT A+B+C       oppure   LET D=A+B+C
                           PRINT D
```

o qualsiasi altra scomposizione in pezzi che porti allo stesso risultato.

A destra dell'uguale, come gia' si e' visto per la frase PRINT, puo' essere posta una espressione aritmetica che a sua volta puo' contenere nomi di

variabili, operazioni e parentesi come visto sopra e come riportato di seguito:

```
LET A4= (12.5-G)*F-897*K
```

Sapete come si fa a incrementare una variabile (cioe' ad aggiungere 1 al valore che gia' contiene)? E' semplicissimo:

```
LET P=P+1
```

significa "prendi il valore di P, aggiungigli 1 e poni il risultato nella cella di nome P". Similmente per:

```
decrementare    LET P=P-1
raddoppiare     LET P=P+P (oppure P*2)
decuplicare     LET P=P*10
ecc.
```

Espressioni serializzate

Abbiamo gia' parlato esaurientemente delle espressioni. In certi casi pero' ci si trova davanti a espressioni del tipo della seguente

```
a+b
--- + (234.8-c)
32
```

Come puo' essere scritta in BASIC questa espressione che occupa piu' di un rigo? Semplice, basta <u>serializzarla</u> ossia scriverla su un rigo solo (cosa che e' <u>sempre</u> possibile), come si fa alla macchina da scrivere e trasformarla cioe' come segue

```
(A+B)/32 + (234.8-C)
```

Il secondo gruppo di parentesi non serve e potrebbe essere eliminato. E' bene pero' tenere presente che le parentesi, benche' rallentino i calcoli non fanno mai male. Nel dubbio e' preferibile usarle quando non servono che non viceversa (senza esagerare, pero!).

Ancora PRINT

La frase PRINT puo' servire anche per stampare piu' dati assieme. Ad esempio

```
PRINT A,34/8,C+D
```

provoca l'uscita sul video di

1) il valore della variabile A

2) il risultato di 34/8 (e' 4.25)

3) il risultato della somma dei valori delle variabili C e D

Le funzioni

Il BASIC, oltre alle quattro operazioni, puo' eseguire un certo numero di funzioni gia' incorporate nel linguaggio. Torneremo in seguito su queste funzioni descrivendole nel dettaglio nell'apposita sezione. Vediamo qui in modo semplice a cosa servono e come si usano. L'esempio piu' banale e' quello della radice quadrata, per ottenere la quale basta scrivere

```
PRINT SQR(25)
```

per avere il risultato 5. Il comando ora visto significa "STAMPA il risultato della radice quadrata di 25". SQR sta per SQUARE ROOT che in inglese significa appunto radice quadrata. Al posto di 25 puo' essere inserita qualsiasi espressione aritmetica, anche con parentesi. Lo stesso risultato di cui sopra lo si poteva ottenere anche dagli esempi che seguono

```
PRINT SQR(20+5)
```

```
LET A=25
PRINT SQR(A)
```

```
LET A=50
PRINT SQR(A/2)
```

ecc.

Le funzioni hanno la priorita' sulla esecuzione delle quattro operazioni anche se ovviamente ven-

gono eseguite solo dopo che l'espressione contenuta tra le parentesi (argomento) e' stata calcolata.

Esempio:

PRINT 5/SQR(20+5)

da' risultato 1 in quanto:

1) Viene eseguita la somma di 20+5.

2) Viene eseguita la radice di 25 (=5).

3) Viene eseguita la divisione 5/5.

Riassunto

Questo capitolo e' molto importante per la compresione dei concetti che seguiranno. E' bene quindi averlo particolarmente chiaro in mente, eventualmente rileggendolo piu' volte per chiarire eventuali punti oscuri. E' molto di piu' che non un mattone su cui costruire le conoscenze successive; e' almeno il 50% di tutto cio' che c'e' da imparare. Quando i suoi contenuti vi saranno chiari, avrete veramente fatto un passo avanti nella vostra conquista dell'informatica.

Abbiamo visto:

1) Che tutte le righe battute devono finire con il tasto RETURN e come correggere eventuali errori di battitura.

2) Come effettuare calcoli anche complicati con la frase PRINT.

3) Cosa e' una espressione aritmetica.

4) Le variabili, come memorizzare dati e risultati intermedi (frase LET) e come usarli nei calcoli successivi.

CAPITOLO III
Sequenze ripetitive

Abbiamo visto nel capitolo precedente che il BASIC dispone di una notevole capacita' aritmetica, puo' cioe' fare i conti come una calcolatrice evoluta ma con alcuni grossi vantaggi:

1) In BASIC le espressioni aritmetiche si scrivono cosi' come si scrivono su carta.

2) Le "memorie", cioe' le variabili, sono molte e possono essere create e richiamate in modo semplice ed intuitivo; la facolta' del programmatore di sceglierne il nome ha una grossa importanza perche' permette di chiamare la variabile con una sigla che ne ricordi il significato: ad esempio P per pi greco, R per raggio, C1 per codice 1 ecc.

3) Il BASIC, a differenza dalla calcolatrice, da' la possibilita' di "commentare" i risultati aggiungendo, nella frase PRINT, delle parole esplicative tra virgolette.

Sequenze noiose

Con i comandi finora visti, nel caso che si debbano eseguire dei conti lunghi e complicati, diventa infinitamente noioso battere sempre i soliti comandi. Nessun problema comunque; il BASIC puo' apprendere una successione di comandi ed eseguirli poi celermente uno dietro l'altro con un comando solo.
Esempio:

Immaginiamo che, dato un prezzo, si debba calcolare lo sconto, il netto, l'IVA ed il totale. Cio'

significa, riportando il problema in termini piu' precisi, calcolare

netto = prezzo - (prezzo x sconto /100)
totale = netto + (netto x aliquota /100)

ossia, se si chiamano

P il prezzo
N il netto
S lo sconto
I l'aliquota IVA
T il totale,

$$N = P - (P \times S / 100)$$
$$T = N + (N \times I / 100).$$

Si osservi che le parentesi non sono necessarie, perche' il BASIC esegue, secondo le buone regole dell'aritmetica, prima le moltiplicazioni e divisioni, quindi le somme e le sottrazioni.

Se il prezzo e' di lire 87600, lo sconto del 12% e l'aliquota IVA del 18%, il problema assegnato puo' essere risolto con le frasi viste nel capitolo precedente cosi':

```
LET P=87600
LET S=12
LET I=18
LET N=P-P*S/100
PRINT "Netto lire ";N
PRINT "Totale lire ";N+N*I/100
```

E' pero' davvero noioso dover ribattere tutte le volte tutti questi comandi. Possiamo memorizzare questa sequenza in modo molto semplice e richiamarla in qualunque momento.

Il numero di linea

Se battiamo un comando facendolo precedere da un numero, il BASIC non lo esegue subito ma lo memorizza. Se battiamo piu' comandi preceduti ciascuno da un numero, il BASIC li memorizza e li pone in ordine, in funzione del numero assegnato loro.

RUN

Il comando RUN (in inglese significa "CORRI") serve poi per comandare l'esecuzione dei comandi

precedentemente memoriazzati, esecuzione che avverra' non secondo l'ordine con cui i comandi sono stati memorizzati, ma nell'ordine indicato dai numeri che abbiamo posto davanti ai comandi stessi.

E' importante osservare che:

1) I numeri da assegnare ai comandi si chiamano numeri di linea.

2) I numeri di linea devono essere compresi tra 1 e 65535 e devono essere interi, cioe' non devono contenere il punto (che in BASIC sostituisce la virgola decimale).

Programma

L'esempio precedente diviene quindi

```
10 LET P=87600
20 LET S=12
30 LET I=18
40 LET N=P-P*S/100
50 PRINT "Netto lire ";N
60 PRINT "Totale lire ";N-N*I/100
```

Una sequenza ordinata di istruzioni BASIC (come ad esempio questa ora vista) si chiama **programma** (visto che non era una cosa poi tanto diffcile?). Il programma indica al BASIC la strada da eseguire per ottenere un certo risultato. Un programma puo' essere semplice come quello ora visto o puo' essere molto complesso e lungo, ma tra i due casi estremi non esiste, da un punto di vista logico, alcuna differenza. In entrambi i casi si tratta di una sequenza di ordini che il BASIC eseguira' fedelmente. L'esecuzione di un programma, come abbiamo visto, la si ottiene con il comando RUN.

Una nota di carattere pratico: si noti nell'esempio sopra riportato che le istruzioni non sono numerate 1, 2, 3... ma 10, 20, 30... Questo perche', nel caso che si debba aggiungere una istruzione che ci siamo dimenticati di battere, non si sia costretti a rifare tutto il lavoro da capo. Volendo aggiungere una istruzione tra l'ultimo LET ed il primo PRINT per stampare il prezzo basta scrivere

```
45 PRINT "Prezzo di listino";P
```

Come abbiamo gia' accennato, il BASIC provvede in modo automatico a mantenere le istruzioni sempre ordinate secondo numeri di linea crescenti e ad inserire quindi la nuova linea tra le due vecchie.

L'esecuzione

Non appena battuto il tasto RETURN dopo il comando RUN, l'operatore perde il controllo della tastiera; il BASIC infatti va al programma a prelevare le istruzioni da eseguire e le esegue fedelmente finche':

1) le istruzioni del programma non sono finite;

2) non venga premuto il RESET o il tasto C mentre si tiene premuto anche il tasto CTRL. Questa pressione combinata di tasti sara' indicata nel seguito CTRL/C e si pronuncia "control ci";

END

3) non si incontri una istruzione STOP (descritta in seguito) oppure una istruzione END. L'istruzione END indica la fine fisica del programma. Pur non essendo sempre indispensabile (come nell'esempio di cui sopra) e' bene inserirla ogni volta per abitudine. Nell'esempio ora visto si aggiungerebbe quindi una

```
70 END.
```

In tutti e tre i casi ora visti si ha l'arresto della esecuzione ed il ritorno ai due punti (:) che indicano che la macchina e' nuovamente in attesa di ordini.

Rivediamo il progamma

LIST

LLIST

L'elenco delle istruzioni che compongono il programma si chiama lista. Per ottenerla basta battere il comando LIST ed essa comparira' subito sul video. E' possibile cosi' ricontrollarla per individuare eventuali errori. Per ottenere la medesima lista sulla carta anziche' sul video si usa il comando LLIST invece di LIST (mi sembra superfluo sottolineare che il comando LLIST

funziona solo a patto che si possegga una stampante e che la medesima, oltre a contenere la carta, sia collegata al computer...).

Come modificare

Il programma puo' essere facilmente modificato in qualsiasi momento, sia per eliminare eventuali errori, che per migliorarlo o apporvi delle aggiunte. Basta:

per sostituire una linea - ribatterla nuovamente. Quando si introduce una linea avente numero uguale ad una gia' esistente, il BASIC sostituisce la nuova alla vecchia

Esempio

```
:LIST                          (battuto dall'operatore)

10 PRINT SQR(49)                     (compare su video)
20 PRINT "Pippo Pluto e Paperino"

:20 PRINT"Il numero sopra e' la radice di 49" (op)
:LIST                                      (operatore)

10 PRINT SQR(49)                     (compare su video)
20 PRINT"Il numero sopra e' la radice di 49"
```

per cancellare una linea - battere il suo numero subito seguito da RETURN. La linea scomparira' dal programma e cessera' di occupare spazio in memoria

Esempio

```
:LIST                          (battuto dall'operatore)

10 LET A=9/6                         (compare su video)
20 LET B=A-4

:20                                        (operatore)
:LIST

10 LET A=9/6                                 (compare)
```

per aggiungere una linea tra altre due - come gia' visto, battere la nuova linea dandole un numero intermedio tra quello precedente e quello seguen-

te.

Esempio

```
:LIST                                   (operatore)

10 LET A=B+C                      (compare su video)
20 LET D=A/B

:15 LET F2=67776/A                      (operatore)
:LIST

10 LET A=B+C                      (compare su video)
15 LET F2=67776/A
20 LET D=A/B
```

ERASE

per cancellare tutto il programma - basta battere il comando ERASE.

Esempio

```
:LIST                                   (operatore)

10 LET A=8                        (compare su video)
20 LET B=56
30 PRINT A/B/78

:ERASE                                  (operatore)
:LIST
```

(non compare niente perche' il programma e' cancellato)

:

Riassunto

La parte sin qui esaminata di questo terzo capitolo e' stato molto importante: ci ha introdotto senza troppa fatica al concetto di programma. Abbiamo infatti visto che:

1) Il BASIC puo' memorizzare dei comandi. La sequenza di questi comandi si chiama programma.

2) Il programma puo' essere eseguito con il comando RUN tutte le volte che sia necessario.

3) Il programma puo' essere modificato con la massima facilita', seguendo le poche regole che abbiamo illustrato, od ampliato o cancellato.

Il grande vantaggio dell'uso dei programmi e' gia' evidente. Procedure di calcolo anche molto complesse possono essere scritte una volta per tutte e poi eseguite quante volte si vuole con dati differenti raggiungendo i risultati in modo piu' sicuro e veloce che non usando la calcolatrice.

Per adesso l'unico modo in cui si puo' ottenere cio' e' quello di modificare via via le frasi LET.

Ingresso da programma

Ma il sostituire di volta in volta i dati modificando la frase LET e' ancora una procedura tediosa. L'ideale sarebbe che fosse lo stesso computer a richiedere i dati diversi di volta in
INPUT volta. Questo lo si ottiene con la frase INPUT.

Esempio:

Supponiamo di dover calcolare la superficie di tre cerchi aventi raggio rispettivamente 12.5, 45.8 e 70.9. Un programma che risolve il problema assegnato potrebbe essere questo:

```
10 LET R=12.5
20 S = R * R * 3.14
30 PRINT "La superficie del cerchio e' ";S
```

Sostituendo la frase 10 con

```
10 LET R=45.8
```

e

```
10 LET R=70.9
```

possiamo calcolare la superficie dei tre cerchi.

Vediamo invece il programma seguente

```
10 INPUT R
```

```
20 S = R * R * 3.14
30 PRINT "La superficie del cerchio e' ";S
```

Quando durante l'esecuzione viene incontrata la frase INPUT (che in inglese significa INGRESSO), l'elaborazione si arresta e sul video compare un punto interrogativo (?) ad indicare che si attende un dato. L'operatore lo batte e quindi preme RETURN; l'elaborazione riprende dopo aver assegnato il valore teste' introdotto alla variabile specificata nella frase INPUT.

Si noti che:

1) Anche INPUT, come PRINT, ammette l'uso di una frase di commento che viene poi riportata prima del punto interrogativo allo scopo di rendere piu' chiaro all'operatore cosa gli venga chiesto. Ad esempio, la frase 10 di cui sopra potrebbe diventare

```
10 INPUT "Raggio del cerchio ";
```

facendo guadagnare notevolmente in chiarezza all'esecuzione del programma medesimo.

2) Una stessa INPUT puo' servire per far accettare piu' variabili; ad esempio

```
10 INPUT A,G,Y7
```

quando questa frase viene incontrata e compare il punto interrogativo si puo'

2a) o battere i tre valori separandoli con una virgola

2b) o battere un solo valore quindi RETURN; la macchina ripropone il punto interrogativo per il secondo e cosi' via finche' non sia stata completata la lista di variabili che segue la parola INPUT.

3) Nella INPUT devono sempre essere dati valori numerici puri e non espressioni aritmetiche. Se si deve battere 18, ad esempio, non e' ammesso battere 12+6 o 20+4-6 ecc.

Chiudendo il capitolo

Abbiamo quindi gia' visto le seguenti frasi BASIC (accanto e' segnato il tipo logico come puntualizzato nella introduzione):

```
INPUT    (ingresso)
LET      (elaborazione)
PRINT    (uscita)
END      (elaborazione)

RUN      (Comando da parte dell'operatore)
CTRL/C   ("       "  "      "     "       )
LIST     ("       "  "      "     "       )
LLIST    ("       "  "      "     "       )
```

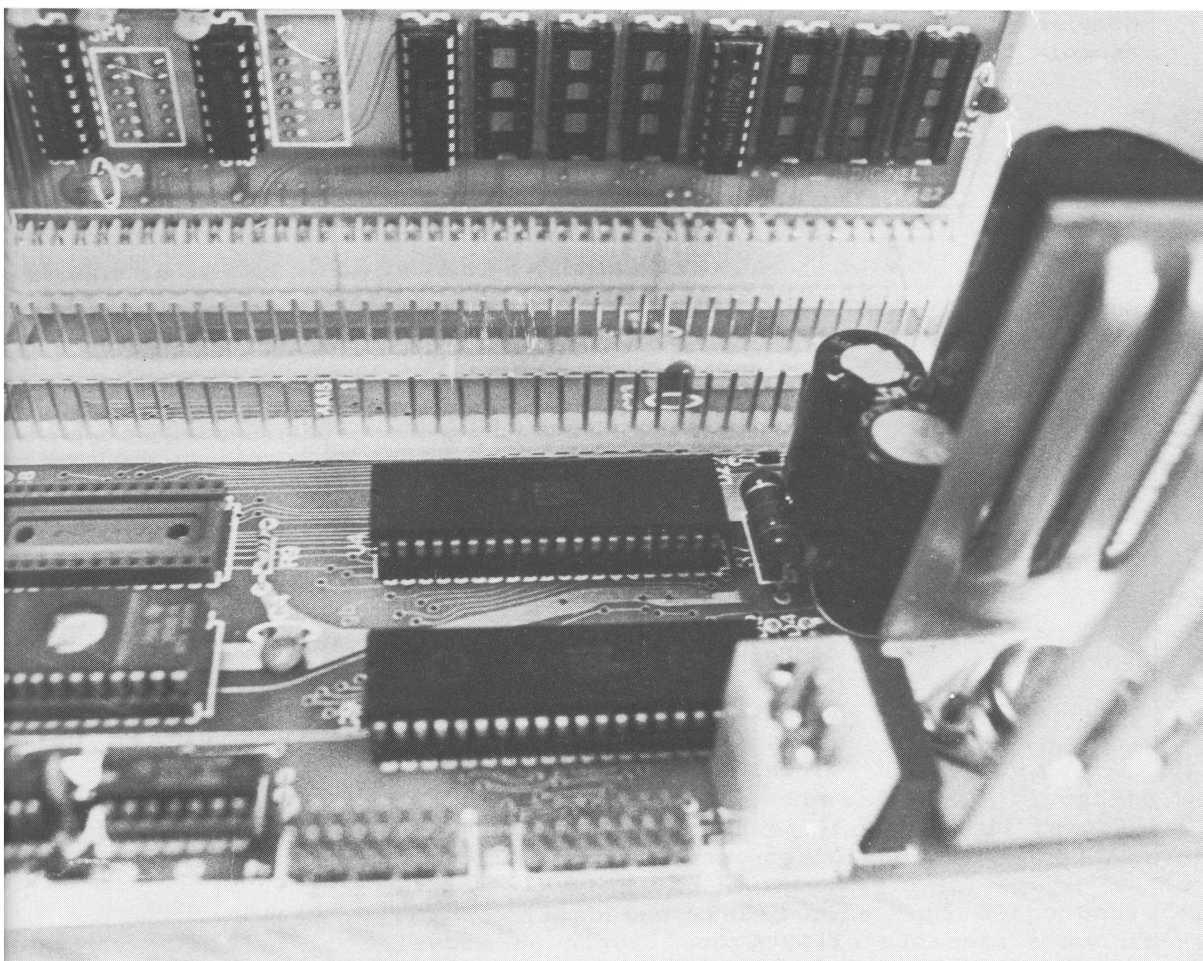



figura 21
Particolare del G5.
Uno di quei "millepiedi" di colore nero è il microprocessore, un moderno componente che racchiude le principali funzioni di un computer e che ha reso possibile tra l'altro la costruzione dei piccoli elaboratori casalinghi.

CAPITOLO IV
Esempi elementari

Semplici esempi

In questo capitolo sono riportati alcuni esempi molto semplici relativi a quanto visto in precedenza. Il lettore li studiera' attentamente e li provera' possibilmente sul suo home computer (gli esempi impiegano tutti istruzioni molto semplici comuni a tutti i BASIC).

Esercizi numerici

1) Calcolare (3x4)+(5x6)+(7*8).

Programma:

```
10 PRINT 3*4 + 5*6 + 7*8
```

Risultato:98

2) Calcolare (3+4)(5+6)(7+8).

Programma:

```
10 PRINT (3+4) * (5+6) * (7*8)
```

Risultato: 1155

3) Calcolare

```
4 x 5     29      19     13 + 3.1415
----- + ------  ----- + -----------
  7     3 x 11  2 + 4         4
```

Programma:

```
10 PRINT (4*5/7+29/(3*11))*(19/(2+4)+((13+
```

```
3.1415)/4))
```

(la linea e' stata divisa per ragioni tipografiche)

Risultato: 20.906

4) Calcolare

```
    1
---------
 1     1
--- + ---
 3     6
```

Programma:

```
10 PRINT 1/(1/3+1/6)
```

Risultato: 2

5) Calcolare

Programma:

```
10 PRINT 3+(1/(7+1/(15+(1/(1+1/292)))))
```

Risultato: 3.1415

Risoluzione di un triangolo

Esaminiamo ora il triangolo riportato nella figura sottostante. Siano noti il lato R e l'angolo t. Si vogliono calcolare i lati x ed y.

Poiche' il nostro GBASIC come ogni altro BASIC lavora su angoli espressi in radianti e non in gradi e' necessario ricordare la semplice formuletta di "passaggio" tra gradi e radianti:

$$\text{angolo in radianti} = \frac{2 \times \text{pi greco}}{360} \times \text{ang.in gradi}$$

La frase 10 pone la variabile P uguale al valore di pi greco.

Programma:

```
10 LET P=3.14159
20 LET X=5 * COS(30 * 2*P/360)
30 LET Y=X * TAN(30 * 2*P/360)
40 PRINT "X = ",X
50 PRINT "Y = ",Y
60 END
```

Risultato: x = 4.33 ; y = 2.49

Svolgimento alternativo: ogni programma puo' in genere essere scritto in diversi modi. Ecco un esempio di una differente stesura dello stesso esempio precedente. Alla frase 30 e' stata fatta una semplificazione: poiche' tan(x)=sin(x)/cos(x) si aveva

$$\cos(x) \cdot \frac{\sin(x)}{\cos(x)}$$

e poiche' moltiplicare per cos(x) e dividere per cos(x) e come non fare proprio niente (attenzione; questo e' un problema di semplice buon senso, non di trigonometria: dividere una torta in tre e moltiplicare la fetta risultante per tre non cambia il peso!).

```
10 LET T = 2 * 3.14159/360 * 30
20 PRINT "X = ", 5 * COS(T)
30 PRINT "Y = ", 5 * SIN(T)
40 END
```

Risposta: la stessa di prima!

Batteri in crescita

Supponiamo di avere una colonia di batteri che si riproducono in continuazione e che accrescono il proprio numero di 1,15 volte al giorno.

Se i batteri sono 1000 al primo giorno, quanti saranno al secondo, terzo, quarto e quinto giorno?

Programma:

```
5 LET F=1.15
10 PRINT "BATTERI INIZIALI 1000"
20 LET G2=1000*F
30 PRINT "SECONDO GIORNO = ",G2
40 LET G3=G2*F
50 PRINT "TERZO GIORNO = ",G3
60 LET G4=G3*F
70 PRINT "QUARTO GIORNO = ",G4
80 LET G5=G4*F
90 PRINT "QUINTO GIORNO = ",G5
100 END
```

```
Risultato: secondo giorno: 1150
           terzo    "     : 1322
           quarto   "     : 1520
           quinto   "     : 1749
```

Conversione pollici

Scrivere un programma che converta da pollici in centimetri (1 pollice = 2,54 cm).

Programma:

```
10 INPUT "POLLICI? ",P
20 PRINT "CORRISPONDONO A CENTIMETRI: ",P*2.54
30 END
```

Volume sfera

Il volume della sfera si calcola moltiplicando pi greco per 4/3 per il raggio della sfera stessa al cubo. Cioe'

$$\text{volume sfera} = \frac{4}{3} \times \text{pi greco} \times \text{raggio}^3$$

Quanto varia il volume di un pallone accrescendone il raggio da 7,5 a 8,25 metri? Si deve ovviamente calcolare il volume delle due sfere, una con raggio 7,5 ed una con raggio 8,25 e valutare poi la differenza di volume.

Programma:

```
10 LET R1=7.5
20 LET R2=8.25
30 LET P=3.14159
40 LET V1 = 4/3 * P * R1*R1*R1
50 LET V2 = 4/3 * P * R2*R2*R2
60 PRINT "DIFFERENZA VOLUMI (MC) ",V2-V1
70 END
```

Risultato: 584.9 metri cubi

Rendite

Supponiamo di avere acquistato un immobile per 140460 dollari e di averlo rivenduto dopo sei mesi e mezzo per 160320 dollari. Se le tasse sottraggono il 35% dell'utile, quale sara' la rendita globale dell'investimento valutata su base annuale?

La formula per il calcolo e' la seguente:

$$\text{rendita (\%)} = \frac{(pv-pa)\ (1-T)}{pa} \ \frac{12}{\text{periodo}} \times 100$$

dove pv = prezzo di vendita = 160320
pa = prezzo di acquisto = 140460
T = tasse = 0.35
periodo = 6.5 mesi

Programma:

```
10 LET T=.35
20 LET P1=140460
30 LET P2=160320
40 LET P3=6.5
50 LET R = (((P2-P1)*(1-T))/P1)*(12/P3)*100
60 PRINT "TASSO ANNUO (%) = ",R
70 END
```

Risultato: 16.96 %

CAPITOLO V
Il BASIC prende decisioni

Decisioni

Abbiamo fin qui visto come il BASIC possa eseguire sequenze anche molto complesse di istruzioni ma non abbiamo ancora imparato a sfruttare veramente la sua potenza. Infatti tutti i programmi visti finora hanno una caratteristica in comune: vengono eseguiti dall'inizio alla fine una sola volta. Ma il BASIC puo' fare di piu': tanto per cominciare, puo' **decidere** se eseguire o meno delle istruzioni.

Esempio

Vogliamo fare un programma che accetti due numeri da tastiera e che, se i numeri sono uguali, scriva "i due numeri sono uguali".

```
10 INPUT "Primo numero ", N1
20 INPUT "Secondo numero ", N2
30 IF N1=N2 THEN PRINT "i due numeri sono uguali"
```

IF

La frase 30 e' nuova: non l'avevamo ancora incontrata. In inglese IF significa SE e THEN significa ALLORA. La frase 30, tradotta, significa quindi: "SE N1 e' uguale a N2 ALLORA STAMPA la frase "i due numeri sono uguali". Quando l'esecuzione del programma arriva alla linea 30, vengono controllate le variabili N1 ed N2; se esse sono uguali viene eseguita la frase che sta dopo la parola THEN, in questo caso PRINT.

Forma generale

La frase IF consente quindi di effettuare un controllo e di prendere delle decisioni di

conseguenza. Al di la' del caso particolare ora visto vediamo in generale come e' costruita:

IF espressione di relazione THEN frase BASIC

L'espressione di relazione e' una espressione, come quella ora vista N1=N2, composta da due e-spressioni aritmetiche poste sui lati di un segno o operatore di relazione, come l'uguale. Ma l'u-guale non e' l'unico operatore di relazione possi-bile. Abbiamo infatti

> (segno di maggiore) - ad esempio N1>N2 si legge "N1 maggiore di N2"

< (segno di minore) - ad esempio N1<N2 si legge "N1 minore di N2"

>= (segno di maggiore od uguale) - ad esempio N1>=N2 si legge "N1 maggiore od uguale ad N2"

<= (segno di minore od uguale) - ad esempio N1<=N2 si legge "N1 minore od uguale ad N2"

<> (segno di diverso) - ad esempio N1<>N2 si legge "N1 diverso da N2"

Frase BASIC e' invece una qualunque frase BASIC, come quelle gia' viste LET, PRINT, INPUT od altre, scritte esattamente allo stesso modo di come le scriveremmo normalmente.

Si noti che abbiamo parlato di "espressioni arit-metiche", da porre sui lati dell'operatore di releazione, quindi non solo nomi di variabili ma anche espressioni piu' complesse, come mostrano i seguenti esempi.

150 IF N1<N2 THEN INPUT N1 - significa: se N1 e' minore di N2 allora accetta da tastiera la varia-bile N1; diversamente prosegui come niente fosse.

16 IF N1/10=((A+B)/(A-B)) THEN LET C=987 - signi-fica: se N1 diviso 10 e' uguale a (A+B)/(A-B) (le parentesi piu' esterne sono superflue) allora poni la variabile C uguale a 987; diversamente prosegui come niente fosse.

130 IF N>100 THEN PRINT"Errore! N troppo grande" - significa: se N e' piu' grande di 100, allora

scrivi "Errore! N troppo grande"; diversamente prosegui come se niente fosse.

30 IF K1+10=V THEN PRINT "BRAVO! Indovinato!" - significa: se K1+10 e' uguale a V allora scrivi "BRAVO Indovinato!"; diversamente prosegui come se niente fosse.

40 IF L<>L1 THEN PRINT "Errore, hai sbagliato!" - significa: se L1 e' diverso da L allora scrivi "Errore, hai sbagliato!"; diversamente prosegui come se niente fosse.

I salti

Come abbiamo gia' sottolineato nell'introduzione di questo capitolo, tutti gli esempi finora visti sono stati relativi a programmi da eseguirsi in rigida sequenza, cioe' una istruzione dopo l'altra. Questa limitazione viene facilmente rimossa per mezzo dell'istruzione GOTO, tradotta VAI A. Con essa e' possibile "saltare" a qualsivoglia numero di linea.

Esempio

```
10 INPUT A
20 PRINT A
30 GOTO 10
```

In questo programma viene accettato un valore A da tastiera, viene stampato e quindi si torna ciclicamente alla frase 10. E' chiaro che una sequenza di questo tipo non finisce mai; per interromperla e' necessario premere CTRL/C o RESET.

IF & GOTO

Per mezzo delle frasi IF e GOTO e' possibile allargare enormemente il campo di azione del BASIC, perche' e' possibile eseguire o non eseguire un certo numero di istruzioni sulla base dei risultati di certe elaborazioni oppure eseguire un certo insieme di istruzioni per un grande numero di volte.

Esempio

Scriviamo il solito programma per calcolare l'area del cerchio; facciamo in modo che l'esecuzione riprenda automaticamente alla fine in seguito ad una interrogazione posta all'operatore:

```
10 INPUT "Raggio?",R
20 LET S=R*R*3.14
30 PRINT "Area del cerchio =",S
40 PRINT
50 PRINT "Per continuare batti 1"
60 PRINT "Per finire qualsiasi altra cosa"
70 INPUT R1
80 IF R1=1 THEN GOTO 10
90 END
```

Il programma e' molto semplice ed e' stato ormai visto piu' volte. Da osservare c'e' solo:

1) La frase 40: PRINT senza niente altro dietro serve per produrre sul video il salto di una riga in piu', per staccare i messaggi successivi.

2) La frase 80: se la risposta alla domanda "Per continuare batti 1" contenuta nella variabile R1 vale 1 allora si esegue la parte dopo la parola THEN, che dice di andare alla linea 10. Diversamente si prosegue come se niente fosse e si incontra la frase END che provoca il ritorno del controllo alla tastiera (compaiono i due punti di attesa comandi).

Un esempio piu' completo

Abbiamo gia' accennato nel capitolo secondo alle funzioni ed abbiamo portato come esempio la funzione SQR() per il calcolo della radice quadrata (SQuare Root in inglese). Facciamo ora la conoscenza di una funzione dal carattere decisamente imprevedibile, RND(0), che genera un numero, compreso tra zero ed uno, del tutto casuale (RND sta per RANDOM, cioe' casuale). La frase

```
10 LET A=RND(0)
```

assegna alla variabile A un valore casuale, variabile di volta in volta, compreso tra 0 ed 1.

Esempio

```
10 PRINT RND(0)
20 PRINT RND(0)
30 END
```

potrebbe dare in esecuzione

```
 .09876
 .97254
```

La funzione RND(0) puo' essere molto utile per i giochi come vedremo in questo esempio che andremo a sviluppare. Vogliamo scrivere un programma che "pensi" un numero e che poi inviti l'operatore ad indovinarlo. Vediamolo

```
10 LET A=RND(0)
20 INPUT "Indovina che numero ho pensato?" ,N
30 IF N=A THEN GOTO 60
40 PRINT "Hai sbagliato, prova ancora"
50 GOTO 20
60 PRINT "Bene! Hai indovinato."
70 INPUT "Un'altra partita?",R
80 IF R=1 THEN GOTO 10
90 END
```

Commentiamolo linea per linea:

10 - Il computer sceglie un numero a caso e lo assegna alla variabile A.

20 - Viene richiesto un tentativo alla tastiera. Il valore accettato viene posto nella variabile N.

30 - Se la risposta e' esatta (N=A), allora si va alla 60. Se no si prosegue.

40 e 50 - Si stampa "Hai sbagliato" e si chiede un nuovo tentativo tornando alla 20.

60 e 70 - Se no si stampa "Bene!" e si chiede se si vuole un'altra partita.

80 - Se la risposta e' affermativa, anziche' tornare alla 20 come nel caso precedente, si torna alla 10 e si assegna ad A un nuovo valora casuale.

Troppo difficle

Il gioco cosi' ottenuto e' pero' troppo difficile: e' praticamente impossibile indovinare

un numero a caso compreso tra 0 e 1!

Modifichiamo il programma in modo che sia esso stesso a facilitarci il compito. Bastera' fargli scrivere "Troppo grande" quando sbagliamo per eccesso e "Troppo piccolo" quando sbagliamo per difetto. Basta aggiungere due "IF"; il programma diviene:

```
10 LET A=RND(0)
20 INPUT "Indovina che numero ho pensato?", N
30 IF N=A THEN GOTO 60
32 IF N>A THEN PRINT "Troppo grande"
34 IF N<A THEN PRINT "Troppo piccolo"
40 PRINT "Hai sbagliato. Prova ancora"
50 GOTO 20
60 PRINT "Bene! Hai indovinato."
70 INPUT "Un'altra partita?",R
80 IF R=1 THEN GOTO 10
90 END
```

Provate, vedrete come e' piu' facile adesso!

Ancora meglio

Il gioco cosi' come e' pero' e' sempre troppo complesso perche' coinvolge l'uso dei decimali, e mal si presta ad esempio per far avvicinare un bambino alla tastiera. Modifichiamolo per farlo lavorare su numeri senza virgola (che in BASIC ricordiamolo e' sostituita dal punto). Per fare questo e' necessario introdurre una nuova funzione: INT(). La funzione INT() ritorna la parte intera di un numero, ossia i numeri che stanno a monte della virgola.

Esempi:

```
PRINT INT(12.876)
12

PRINT INT(0.9876)
0

PRINT INT(1234)
1234

PRINT INT(.897+123)
123
```

```
PRINT INT(A1)
34

PRINT INT(A1+5)
39
```

Se applichiamo la funzione INT alla funzione RND(0) ora vista scrivendo

```
10 LET A=INT(RND(0))
```

avremo pero' come ovvio sempre risultati a valore zero; infatti la RND(0) da' un numero compreso tra zero ed uno senza quindi nessuna cifra prima della virgola (punto) decimale. Se vogliamo dei numeri interi e' quindi necessario moltiplicare il numero ottenuto per dieci o cento o quello che si vuole. La 10 diviene quindi

```
10 LET A=INT(100*RND(0))
```

Moltiplicare per cento e' come spostare la virgola (punto) di due punti a destra; si ottengono quindi numeri casuali interi di due cifre. Desiderandoli di tre basta moltiplicare per mille, di 4 per diecimila e cosi' via.

Se desideriamo che i numeri risultanti non siano maggiori ad esempio di 255 basta moltiplicare invece per 255 (tenete a mente questa osservazione; ci sara' utile quando parleremo della grafica).

Si noti che quello ora visto e' l'esempio piu' completo sin qui trattato. Meditatelo attentamente, perche' in esso sono contenuti un po' tutti i concetti che stanno alla base anche dei programmi piu' complessi.

Riassunto

Questo capitolo e' stato dedicato quindi ad esaminare:

1) La frase IF nella sua forma generale e, con essa collegate, le espressioni di relazione ed i relativi operatori (=, >, < ecc.). Con la frase IF..THEN.. (che, ricordiamolo significa SE..ALLO-

RA..) il programma BASIC puo' assumere delle decisioni in funzione dei valori di certe quantita'.

2) La frase GOTO, usata per eseguire dei salti da una istruzone ad un'altra e per alterare quindi l'ordine di esecuzione che avverrebbe diversamente in stretta sequenza.

Occasionalmente, per sviluppare meglio l'esempio, abbiamo anche visto le funzioni:

RND(0) che ritorna un numero casuale compreso tra zero ed uno.

INT() che ritorna la parte intera di un numero (di un valore assegnato, di una variabile, di una espressione).

Super-riassunto

Vediamo quindi di dare un occhiata piu' dall'alto sui capitoli fin qui scorsi:

CAPITOLO I - concetti generali

CAPITOLO II - il BASIC a simulare una super calcolatrice; i concetti di variabile e di espressione aritmetica.

CAPITOLO III - la programmabilita' del BASIC. Comandi ausiliari per creare, listare e modificare i programmi.

CAPITOLO IV - esempi sui concetti visti.

CAPITOLO V - il BASIC e le decisioni (frase IF..THEN..). Come alterare l'ordine di esecuzione delle istruzioni (frase GOTO). Il primo esempio piu' complesso.

CAPITOLO VI
Cicli ripetitivi

Cicli ripetitivi

Supponiamo di dover realizzare dei programmi dove una serie di istruzioni viene ciclicamente ripetuta piu' volte. Ad esempio sia necessario fare un programma che prepara una tabella con i costi orari di vari elettrodomestici in funzione della potenza assorbita. Supponiamo il costo di un wattora pari a 0.178 lire e di voler creare una tabella che dia tutti i valori di costo per assorbimenti compresi tra 1000 e 2000 watt a passi di 100W. Per risolvere questo problema si deve semplicemente fare una serie di moltiplicazioni:

```
1000 x 0,178 =
1100 x 0,178 =
1200 x 0,178 =
1300 x 0,178 =
           ...... ecc.
```

aggiungendo 100 di volta in volta fino a raggiungere 2000. Il BASIC dispone di una istruzione fatta apposta a questo scopo: la FOR..STEP. Esempio

```
10 FOR W=1000 TO 2000 STEP 100
20 PRINT W * .178
30 NEXT
```

L'istruzione alla linea 10 dice: per W che va da 1000 a 2000 con passo di 100 esegui tutte le istruzioni che stanno da qui alla prima frase NEXT che incontri (FOR significa PER, TO significa A, STEP significa PASSO, NEXT significa PROSSIMA). Quindi il ciclo compreso tra l'istruzione FOR e la prima NEXT che segue viene eseguito una prima volta per W=1000, poi per W=1200, cioe' per W piu' il passo indicato (che e' di 100 nell'esempio) e

cosi' via. Quando si raggiunge la NEXT si fa il controllo se W e' giunto al limite superiore (quello cioe' indicato dopo il TO). In caso negativo si torna alla FOR, si incrementa W del PASSO indicato e cosi' via. In caso affermativo, cioe' se W ha raggiunto il limite superiore, si prosegue.

Maggiori informazioni

Se non si scrive STEP.., si assume automaticamente il passo di 1. Sia gli estremi di variazione della variabile che il passo possono essere risultati di espressioni aritmetiche:

Esempio

```
10 FOR I=K-5 TO K+5 STEP J/8
.....
60 NEXT
```

significa: per I che va dal valore ottenuto sottraendo 5 alla variabile K (K deve essere necessariamente gia' definita, con una LET o una INPUT!) al valore ottenuto sommando 5 a K e con passo dato da J diviso per 8 (anche J deve essere ovviamente definito), esegui le istruzioni tra qui e la prossima NEXT.

Il passo puo' essere anche negativo e quindi procedere all'indietro.

Esempio

```
10 FOR F=10 TO 0 STEP -1
 ...
20 NEXT
```

significa: per F che va all'indietro da 10 a 0 con passo -1.......

Si possono mettere piu' FOR uno dentro l'altro:

```
10 FOR I=1 TO 10
20 FOR K=20 TO 30 STEP 2
30 ......
40 ......
50 NEXT
60 ......
```

```
70 ......
80 NEXT
```

Il primo ciclo inizia con la 10 e si chiude con la 80; il secondo inizia con la 20 e finisce con la 50. Il primo ciclo viene eseguito 10 volte (I che va da 1 a 10). Il secondo, quello "innestato" nel primo, viene eseguito 5 volte (K da 20 a 30 passo 2) ogni volta che si esegue il primo, in totale 50 volte.

Utilita'del FOR

L'istruzione FOR e' molto usata, come vedremo in seguito, in una grande varieta' di casi. La sua flessibilita' applicativa e' notevole, tuttavia presenta una sostanziale differenza rispetto alle istruzioni che abbiamo prima imparato ad usare: non e' indispensabile. Infatti la sequenza FOR..TO..STEP..NEXT puo' essere benissimo simulata usando invece la IF..THEN ed il GOTO. Esaminaimo infatti questi blocchi di istruzioni che sono equivalenti:

```
10 FOR I=1 TO 100
20 PRINT "VIVA IL BASIC!"
30 NEXT
40 END
```

```
10 LET I=1
20 PRINT "VIVA IL BASIC"
30 I=I+1
40 IF I<=100 THEN GOTO 20
50 END
```

Il primo e' talmente ovvio che non ci spendiamo sopra alcun commento; si limita a stampare 100 volte di seguito la frase "VIVA IL BASIC". Anche il secondo lo fa, senza la frase FOR, ma in forma meno immediata e quindi meno intuitiva. Analizziamolo frase per frase:

10 Inizializza la variabile I al valore 1.

20 Stampa

30 <u>Incrementa</u> (ricordate che lo avevamo gia' visto?) la variabile I aggiungendo alla I stessa la quantita' 1; in altre parole noi diciamo all'ela-

boratore: "poni il contenuto della variabile I uguale al valore di I piu' uno".

40 Controllo: se I e' minore od uguale a 100 si torna alla frase 20, in caso contrario abbiamo finito.

Si osservi che la frase FOR consente non solo di risparmiare una istruzione, ma anche una migliore "leggibilita'" del programma, cosa questa molto vantaggiosa quando si debba rimetterci le mani a distanza di tempo (scoprirete come e' facile dimenticare!).

Le spiegazioni

Visto che siamo in argomento, soffermiamoci un po' su questo aspetto ora visto della "legibilita'". Avrete certo gia' scoperto per conto vostro che le frasi BASIC sono dei "mattoncini" molto ben congegnati con le quali e' facile costruire programmi abbastanza complicati senza nessuna difficolta'. E' un po' come quando si impara ad andare in bicicletta, dapprima sembra difficilissimo se non quasi impossibile, successivamente si scopre quanto sia semplice e si comincia a scorrazzare avanti e indietro. I programmi complicati non sono quindi appannaggio dei super esperti. Una volta capito il trucco si cominciano a mettere insieme i mattoncini e ci si accorge che si possono costruire dei grattacieli. I guai sorgono invece in due circostanze:

1) Quando si debbano "rimettere le mani" in un programma a distanza di tempo. Le cose che mentre scrivevamo le frasi sembravano ovvie e scontate diventano misteriose ed oscure; "perche' avro' fatto cosi'?" ci si chiede.

2) Quando siano delle terze persone a dover capire nella lista del nostro programma.

REM

Per limitare gli effetti di questo inconveviente e' stata inventata la frase REM che sta per REMARK, cioe' COMMENTO. Tutto cio' che sta sul rigo dopo la parola REM viene completamente ignorato dal BASIC anche se e' riportato nella lista e viene usato dal programmatore per le annotazioni.

Esempio

```
10 REM PROGRAMMA PER IL CALCOLO AREA TRIANGOLO
20 REM
50 REM INGRESSO DATI
40 INPUT "Base? ",B
50 INPUT "Altezza? ",A
60 REM ELABORAZIONE DEL RISULTATO
70 LET S=B*A/2
80 REM USCITA RISULTATI
90 PRINT "Area del tringolo=",S
100 END
```

Tutte le frasi REM, pur occupando memoria, sono ignorate dal BASIC ma servono per rendere il programma piu' leggibile". La frase 20 e' usata solo allo scopo di staccare il titolo dal resto.

I principianti

I principianti tendono sempre a sottovalutare l'enorme importanza dei commenti e ne sono percio' generalmente avari, anche se pagano poi nel tempo questa leggerezza e cambiano idea.... E' chiaro che i commenti devono comunque essere apposti con una certa saggezza, evitando le spiegazioni inutili e riducendoli all'essenziale, sia per evitare sprechi di memoria, sia per non rallentare eccessivamente il programma.

Riassunto

In un certo senso questo e' stato un capitolo di riposo, almeno dal punto di vista dei concetti importanti. Abbiamo esaminato la frase FOR dicendo he non e' una frase indispensabile, in quanto diversamente "surrogabile", ma molto pratica e di uso assai frequente. Abbiamo poi parlato dei commenti, frase REM, e della loro importanza.

Col prossimo capitolo invece riprenderemo a procedere sui concetti di rilievo, affrontando il problema dei <u>sottoprogrammi</u>.

CAPITOLO VII
I sottoprogrammi

Sezioni di uso frequente

Capita frequentemente nello scrivere dei programmi di avere un'insieme di istruzioni di uso ricorrente. Ad esempio, supponiamo di avere un programma in cui vengano richiesti spesso dei dati da tastiera e di voler controllare questi dati per vedere se rientrano in certi limiti assegnati. Piu' specificamente, dobbiamo accettare tre variabili, X, Y e Z e dobbiamo controllare che

X sia compreso tra 0 e 255 (si scrive 0<X<255, cioe' X deve essere maggiore (>) di 0 e minore (<) di 255).

Y deve essere compreso tra 0 e 191 (0<Y<191).

Z deve essere compreso tra 1 e 16.

Nel caso in cui uno dei valori introdotti non rispetti questi concetti, si deve emettere un messaggio di avviso "Dato errato - ribattere tutto" e richiedere di nuovo i tre dati. Sulla base delle informazione fin qui fornite e' veramente facile scrivere un programma siffatto.

Esempio:

```
10 INPUT "Introduci X, Y e Z ",X,Y,Z
20 IF X<0 THEN GOTO 100
30 IF Y<0 THEN GOTO 100
40 IF Z<1 THEN GOTO 100
50 IF X>255 THEN GOTO 100
60 IF Y>191 THEN GOTO 100
70 IF Z>16 THEN GOTO 100
.... (il programma continua usando i dati letti)
100 PRINT "Dato errato - ribattere tutto"
```

```
110 GOTO 10
```

Se nel corso del programma che utilizza X, Y e Z e' necessario fare piu' accettazioni da tastiera dei dati medesimi, e' chiaro ed evidente che per ciascuna di esse, per avere la convalida dei dati introdotti, sia necessario ripetere tutte le istruzioni sopra viste, con grande spreco di memoria e con maggiore complessita' di scrittura, quindi con minore leggibilita'.

I sottoprogrammi

I sottoprogrammi servono per aggirare questo inconveniente. I sottoprogrammi sono delle parti di programma richiamabili da parte di una istruzione, detta GOSUB, che passa il controllo ad una linea indicata. L'esecuzione continua cosi' nel sottoprogramma esattamente come per la GOTO, ma con una grossa e sostanziale differenza; che mentre nella GOTO una volta effettuato il salto si continua in sequenza, nella GOSUB, dopo fatto il salto, si continua in sequenza finche' non si incontra una istruzione RETURN, alla quale si torna alla istruzione subito seguente alla GOSUB.

Esempio:

Per trasformare in sottoprogramma il programma precedentemente esaminato di accettazione dati da tastiera basta aggiungere una sola istruzione

```
80 RETURN
```

in questo modo tutte le istruzioni ora viste possono essere richiamate in qualunque momento ed in qualsiasi posto del programma con una istruzione GOSUB 10.

Ad esempio

```
1000 REM ACCETTAZIONE DATI PUNTO PARTENZA
1010 REM
1020 GOSUB 10
1030 REM CALCOLO DEL PUNTO DI PARTENZA
1040 M=(X+Y+Z)/.98763
1050 REM ACCETTAZIONE DATI PUNTO ARRIVO
1060 GOSUB 10
1070 REM CALCOLO DEL PUNTO DI ARRIVO
```

```
1080 M1=(X+Y+Z)/.98763
 .....
```

Vantaggi

I vantaggi dell'uso dei sottoprogrammi sono molteplici; infatti, oltre a quello piu' ovvio di risparmiare memoria e fatica, ne hanno un altro forse addirittura piu' importante: quello di rendere piu' semplice la struttura di un programma. Questo vantaggio e' tanto grande che spesso si creano dei sottoprogrammi anche da eseguire una sola volta, solo per "snellire" la costruzione di una procedura complicata. Ad esempio, un programma potrebbe essere cosi' composto

```
10 REM ACCETTAZIONE DATI
20 GOSUB 1000
30 REM CALCOLO PREZZI
40 GOSUB 2000
50 REM STAMPA FATTURE
60 GOSUB 3000
.....
```

In questo modo e' facile spezzare un programma complesso in tanti "moduli", i quali a loro volta possono essere composti di altri moduli e cosi' via fino a spezzettare la complessa procedura in tante piccole parti relativamente semplici, ottenendo anche l'importante vantaggio di una piu' facile modificabilita'.

CAPITOLO VIII
Variabili multiple

Variabili multiple

Alle volte puo' essere conveniente identificare piu' variabili con nomi simili. Ad esempio, supponiamo di voler memorizzare gli incassi di un negozio durante tutto il mese. Seguendo il modo di procedere sin qui esaminato, dovremmo memorizzare ciascun dato in una variabile differente; ad esempio I1 per il giorno 1, I2 per il giorno 2 ecc. A parte il non trascurabile inconveniente che i giorni sono 31 mentre le nostre variabili arrivano al piu' a I9 (ignoriamo per ora questo problema) il BASIC consente di fare meglio: consente di definire una sola variabile per cosi' dire "multipla" che si chiama sempre I e viene seguita da un numero, in parentesi, che dice a quale elemento specifico intendiamo riferirci. Quindi invece di I1 abbiamo I(1), invece di I2 abbiamo I(2) e cosi' via.

I(0)	I(1)	I(2)	I(3)	I(4)	I(5)	---

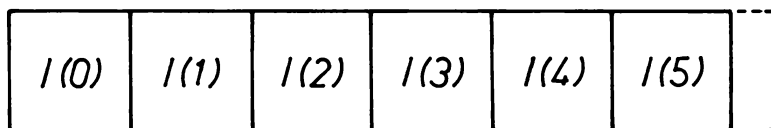



figura 22

Attenzione: non e' una differenza solo formale perche' (questo e' il **grosso** vantaggio) il numero che sta tra parentesi puo' essere anche una espressione aritmetica; sono quindi ammesse scritture del tipo

```
I(3)
I(3+K)
I(K*3)
```

ecc.

L'utile che ne consegue e' molto grande, come mostra l'esempio che segue.

Esempio

Riferendosi a quanto detto, supponiamo di avere memorizzati in I1, I2 ecc. gli incassi della settimana e di voler fare un programma che su richiesta mostri su video il totale desiderato. Senza le variabili ad indice ora viste, avremmo

```
10 INPUT "Quale giorno desideri", G
20 IF G=1 THEN PRINT I1
40 IF G=2 THEN PRINT I2
50 IF G=3 THEN PRINT I3
60 IF G=4 THEN PRINT I4
70 IF G=5 THEN PRINT I5
80 IF G=6 THEN PRINT I6
90 IF G=7 THEN PRINT I7
..... ecc.
```

Con le variabili ad indice il programma ora visto si semplifica drasticamente:

```
10 INPUT "Quale giorno desideri? ",G
20 PRINT I(G)
 .... ecc.
```

L'ultima frase 20 dice: stampa l'elemento G della variabile I. Se G vale 1 stampa l'elemento 1, se G vale 2, l'elemento 2 e cosi' via. Notiamo per inciso che in ambo i casi e' opportuno fare un controllo sulla variabile G introdotta, ad esempio aggiungendo due frasi

```
15 IF G<1 THEN GOTO 10
17 IF G>7 THEN GOTO 10
```

in questo modo l'introduzione di un numero errato provoca la ripetizione della domanda.

Molti elementi

Come gia' evidenziato le variabili ad indice sono preziose anche perche' mettono a disposizione molti piu' elementi rispetto alle variabili comuni. Se infatti invece di giorni della settimana si fosse dovuto memorizzare i giorni del mese saremmo stati in un bel guaio; arrivati a I9 non avremmo saputo come andare avanti e saremmo stati costretti ad usare un'altra variabile e cosi' via: una vera complicazione. Con le variabili ad indice non ci sono problemi; basta solo aggiungere all'inizio del programma una frase DIM (DIM sta per DIMENSION, DIMENSIONE) dove sia indicato il massimo numero di elementi per quella variabile. Esempio, se vogliamo che la variabile I sia composta di elementi da 1 a 31 dobbiamo scrivere

DIM

```
10 DIM I(31)
```

Negli esempi precedenti non abbiamo usato la frase DIM, perche' il BASIC dimensiona automaticamente le variabili con indice minore di 10. Se cioe' usiamo indici minori di 10 non e' necessaria la frase DIM, oltre si.

Si noti poi che la frase ora vista DIM I(31) genera una variabile multipla a **32** e non 31 elementi; questo perche' le variabili multiple **hanno sempre l'elemento a indice 0** (cioe' I(0).

Nel GBASIC il numero di elementi di una variabile e' limitato in pratica dalle dimensioni della memoria.

CAPITOLO IX
La grafica elementare

Ed eccoci finalmente giunti alla **grafica** che costituisce certo la parte piu' attraente perche' consente di ottenere una grande varieta' di effetti. Il BASIC originale, quando fu concepito, non prevedeva la grafica in alcun modo poiche' era destinato per lo piu' a funzionare tramite una telescrivente piuttosto primitiva. Successivamente, in particolare con l'avvento del personal computer, furono aggiunte delle frasi adatte a governare le nuove possibilita' dello schermo anche se i vari progettisti, operando separatamente, non hanno seguito esattamente le stesse linee di azione. La standardizzazione in questo senso e' stata pertanto inferiore. Il G5 e' dotato di un set particolarmente ricco di funzioni grafiche, studiato con grande attenzione affinche' fosse logico, intuibile e facile da usare. Passando ad altri microcomputer puo' essere necessario ricorrere a qualche giro un po' piu' complesso, in particolare sugli sprites, per ottenere gli stessi risultati che sul G5 possono essere conseguiti con poche istruzioni.

Grafica: applicazioni illimitate

La grafica e' veramente affascinante. Gia' molto tempo prima dell'invenzione del computer, il grande Leonardo osservava che "vale molto piu' un buon disegno di tante parole". L'uso della grafica, col BASIC, e' facilissimo e consente una infinita varieta' di applicazioni in ogni campo. Ad esempio

Nel campo **tecnico scientifico**, per ottenere tracciati di funzioni, diagrammi, disegni di parti

meccaniche ecc. Questa caratteristica e' amplificata dalla possibilita' di interfacciarsi direttamente con strumentazione, per cui i dati letti possono essere visualizzati direttamente in forma grafica.

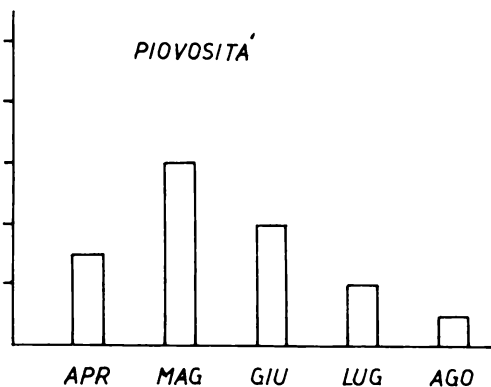
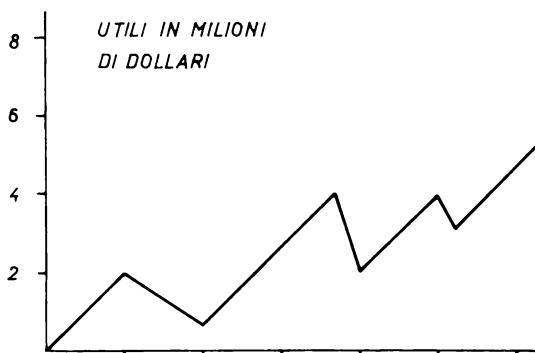
figura 23

Nel campo **finanziario:** una serie di diagrammi opportunamente tracciati evidenziano situazioni meglio di tanti tabulati numerici. Pensate ad esempio al classico profilo dei profitti per periodo sempre presente nell'ufficio di Paperon de' Paperoni:

figura 24

Nel campo **dei giochi:** solo l'immaginazione pone un limite alle infinite applicazioni di un computer dotato di grafica nel campo dei giochi, specie quando, come il G5, possieda anche delle facilita' per l'animazione. Guerre spaziali, ping-pong, pallone, battaglie, simulazioni ecc. ecc. Buona parte degli esempi saranno dedicati appunto ai giochi.

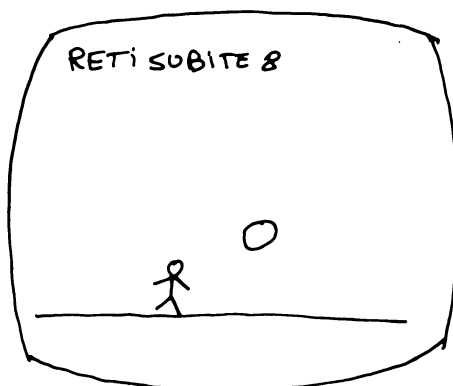



figura 25

Nel campo **pubblicistico.** Un microcomputer dotato di grafica puo' essere impiegato per applicazioni pubblicitarie, illustrative o didattiche con possibilita'di ottenere una elevata resa nel processo di comunicazione.

figura 26

Ma i possibili impieghi della grafica non si fermano qui. Se le applicazioni del BASIC sono svariate quelle del BASIC combinato con la grafica sono certo infinite. Nel seguito ne esamineremo una piccolissima parte ma da esse per il lettore sara' facile, aiutandosi con la fantasia, ottenere quelle di suo particolare interesse.

Lo schermo

All'accensione lo schermo del G5 e' posto in modo TEXT. Sul video trovano posto 24 righe di 40 caratteri cadauna e si ha lo "scroll automatico", ossia quando lo schermo e' pieno le righe scorrono di un posto verso l'alto; la prima riga scompare e viene perduta, mentre l'ultima diventa

penultima. La ventiquattresima riga si pulisce e serve per la scrittura successiva e cosi' di volta in volta, ne' piu' ne' meno del foglio nella macchina per scrivere.

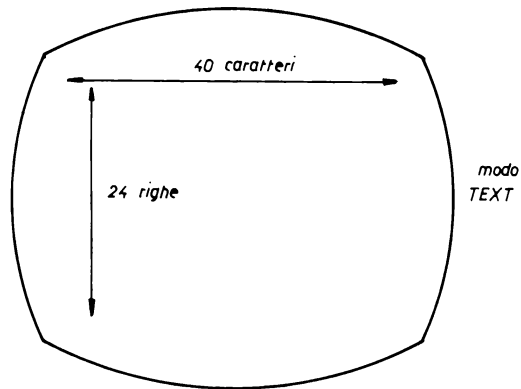



figura 27

GRAPH

Con il comando GRAPH (che puo' essere usato anche in una linea di programma) si passa invece nel modo grafico (o GRAPH). Le linee di testo sono sempre 24 ma di soli 32 caratteri, che risultano quindi piu' spaziati rispetto al modo TEXT. Non si ha piu' lo scroll automatico. Quando si giunge a scrivere in fondo alla pagina tutte le scritture avvengono ripetutamente sull'ultima riga e l'immagine resta fissa.

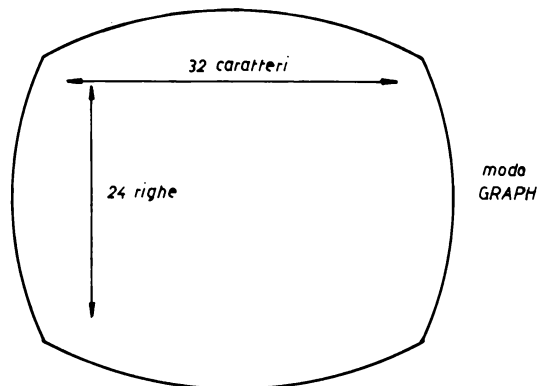



figura 28

AT

E' possibile scrivere tuttavia in qualunque posizione del video facendo precedere l'istruzione PRINT o INPUT dalla frase AT seguita dal numero di riga e di colonna dove deve avvenire la scrittura. Ad esempio

```
10 AT 20,10
```

```
20 PRINT "Sono a riga 20 e colonna 10"
30 END
```

provoca la scrittura della frase "Sono a colonna..." a partire dalla riga 20 e dalla colonna 10. Tutte le frasi PRINT successive provocheranno quindi uscite sulle righe seguenti 11, 12 ecc. fino alla riga 24. Dopo ancora tutte le frasi PRINT provocheranno l'uscita sulla riga 24.

NOTA - ricordarsi sempre che la prima riga e' la riga 0 e che la prima colonna e' la colonna 0.

Tutto cio' per quanto riguarda le scritte alfanumeriche. Nel modo GRAPH tuttavia lo schermo viene idealmente diviso in un reticolo largo 256 punti ed alto 192:

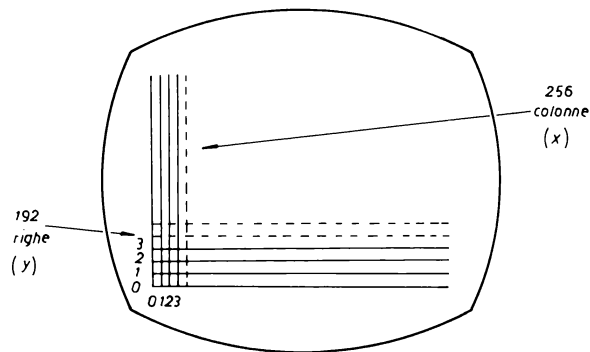



figura 29

Esattamente come nella battaglia navale, qualsiasi punto puo' essere identificato con una coppia di numeri; il primo che corrisponde alla sua posizione orizzontale (chiamata anche "x" o "ascissa") ed il secondo che corrisponde alla sua posizione verticale (chiamata anche "y" o "ordinata"). Con il BASIC del G5 e' possibile accendere o spegnere qualsiasi punto sullo schermo in modo molto semplice. Basta battere

SET-RESET

```
SET 10,75
```

ed il punto di coordinate 10,75 si illuminera' sul video.

Per spegnerlo basta invece

```
RESET 10,75
```

Sia SET che RESET possono essere ovviamente usate come istruzioni di un programma mentre i numeri 10 e 75 possono essere delle espressioni aritmetiche.

Un esempio

Vediamo subito un semplice programma esempio la cui esecuzione chiarira' subito ogni dubbio residuo:

```
10 INPUT "Quale punto devo accendere ? ", X,Y
20 SET X,Y
30 GOTO 10
40 END
```

Questo programma esegue le seguenti operazioni:

1) Chiede le coordinate del punto da accendere sul video

2) accende il punto richiesto

3) continua finche' non interrotto con il CTRL/C o col RESET.

Provate per esercizio a scrivere un programma che domandi se si desidera fare una accensione o uno spegnimento ed utilizzi poi l'istruzione SET o RESET di conseguenza.

Tracciamo un segmento

Avrete gia' intuito a questo punto che qualsiasi forma che possa comparire sul video e' ottenuta provocando l'accensione dei singoli punti del reticolo prima descritto. Una retta orizzontale, ad esempio, la si ottiene illuminando una serie di punti consecutivi con la y mantenuta costante (all'inverso una retta verticale):

Per fare cio' basta un programma molto semplice:

```
10 LET Y=50
20 FOR X=100 TO 200
30 SET X,Y
40 NEXT
50 END
```

figura 30

In questo modo si illuminano tutti i punti di coordinate

100,50
101,50
102,50
....
200,50

e si traccia quindi un segmento, a y=50, lungo 100 punti (da x=100 a x=200). Si osservi che negli esempi si usa spesso identificre le variabili x e y con X e Y. Cio' e' fatto per chiarezza ma niente proibisce di chiamarle in altro modo (A e B, W3 e R ecc. ecc.).

Ancora sui modi TEXT

Quando si e' in modo grafico si puo' tornare nel modo TEXT con il comando TEXT, che puo' essere anche usato come linea di programma. Sia TEXT che GRAPH (che, lo ricordiamo, serve per passare in modo grafico), provocano la completa ripulitura del video ed il posizionamento delle successive scritture nell'angolo in alto a sinistra. Possono essere quindi anche utilizzati senza cambiare modo ma solo per azzerare il video. Ad esempio, se siamo in modo TEXT e vogliamo ripulire lo schermo basta impartire il comando TEXT.

Nel modo TEXT la posizione operativa viene segnalata dal cursore, un piccolo trattino di sottolineatura che indica dove avverra' la prossima scrittura. Nel modo GRAPH il cursore non e' pre-

sente.

Al comando GRAPH, l'origine delle coordinate e' posizionata nell'angolo in basso a sinistra (vedi paragrafo seguente).

L'origine

Il punto di coordinate 0,0 e' detto origine delle coordinate o degli assi o semplicemente origine. Al comando GRAPH l'origine e' posta automaticamente nell'angolo in basso a sinistra. In certe circostanze tuttavia puo' essere utile "spostare" l'origine in un punto diverso, come ad esempio, caso frequente, al centro dello schermo. Questo puo' essere fatto in modo molto semplice con il comando ORG che puo' apparire anche come frase di programma e che deve essere seguito dalle coordinate della nuova origine, sempre riferita al sistema di coordinate iniziali, cioe' quelle aventi il punto 0,0 nell'angolo in basso a sinistra. Per avere l'origine al centro basta battere

```
ORG 127,95
```

(non 128,96, perche' il primo punto e' il punto 0 e non 1).

Dopo l'istruzione ora vista, la frase

```
SET 0,0
```

provoca l'illuminarsi non del punto in basso a sinistra ma del punto al centro dello schermo. Le coordinate di tutti gli altri punti sono alterate di conseguenza.

Lo spostamento dell'origine da' luogo come ovvio a coordinate anche negative, come riportato nella sottostante figura a pagina seguente.

Si osservi inoltre che l'istruzione ORG non provoca nessuna alterazione di una immagine gia' esistente sullo schermo ma viene semplicemente usata come indicazione per le operazioni successive.

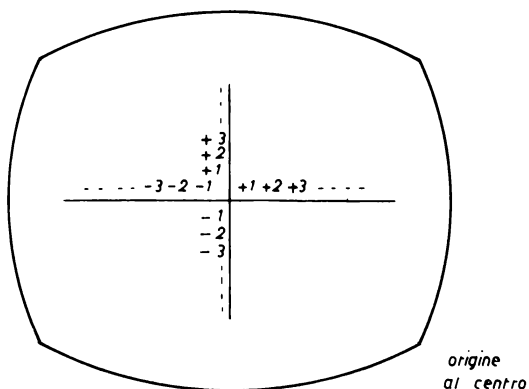



figura 31

Errori

Il GBASIC, nei comandi che indicano delle coordinate, esegue dei controlli e segnala errori quando il comando e' incorretto (errore di linguaggio) o quando una o piu' espressioni aritmetiche hanno un risultato maggiore di 255 (errore di argomento). Cio' potrebbe apparire curioso: infatti l'errore dovrebbe comparire anche quando la y e' maggiore di 191. Questo non e' vero, perche' il G5 si comporta sempre come se l'area visualizzata, di 192x256 punti, fosse una porzione di un'area piu' grande di 256x256 punti. E' quindi lecito effettuare operazioni anche nella parte non visibile che sta al di sopra dello schermo.

Riassunto

Abbiamo quindi finora esaminato:

1) L'organizzazione del video.

2) I due modi di funzionamento TEXT e GRAPH.

3) La frase AT per predisporre la scrittura alfanumerica in un certo punto dello schermo.

4) Le frasi SET/RESET per accendere/spegnere un punto a piacere sullo schermo.

5) La frase ORG per muovere l'origine degli assi.

PLOT

La coppia SET/RESET rappresenta un modo elementare di accedere alla grafica; con esse

potenzialmente (in teoria) si puo' fare tutto. Abbiamo gia' visto un modo di tracciare di una retta orizzontale. Il descrivere rette tra due punti qualsiasi e' pero' un problema assai piu' complesso, specie quando l'origine degli assi e' spostata e ci sono anche coordinate negative. Ma il GBASIC ci viene ancora una volta in aiuto con le frasi PLOT/UNPLOT (in italiano disegna/scancella) utili per ogni tipo di tracciamento. Scrivendo

```
PLOT 10,0,20,70
```

si provoca (ovviamente sempre in modo GRAPH) il comparire di un segmento tra i punti di coordinate 10,0 e 20,70. I numeri 10,0,20 e 70 possono essere sostituiti da espressioni aritmetiche. UNPLOT compie esattamente lo stesso lavoro di PLOT solo invertito, cioe' cancellando anziche' disegnando.

La frase PLOT e' certo la piu' versatile e potente istruzione per l'uso della grafica come vedremo nei numerosi esempi.

Riquadriamo lo schermo

Per iniziare a prendere dimestichezza con la frase PLOT cominciamo con il riquadrare lo schermo video, ossia con il tracciare una linea continua attorno allo schermo sul margine estremo. In pratica si devono solo tracciare 4 segmenti tra i punti di coordinate:

da	a
0,0	255,0
255,0	255,191
255,191	0,191
0,191	0,0

Il programma richiesto e' quindi il seguente:

```
10 PLOT 0,0,255,0
20 PLOT 255,0,255,191
30 PLOT 255,191,0,191
40 PLOT 0,191,0,0
50 END
```

Se siamo in modo TEXT e' opportuno aggiungere una

istruzione

```
5 GRAPH
```

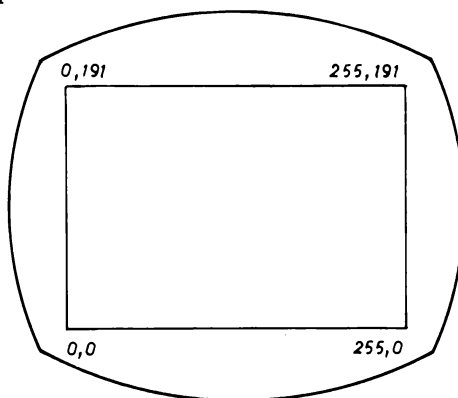



figura 32

Le diagonali

Per avere anche il tracciamento delle diagonali che dividono la pagina video basta aggiungere solo le due istruzioni che servono per unire i punti 0,0 con 255,191 e 0,191 con 255,0. Basta quindi aggiungere

```
50 PLOT 0,0,255,191
60 PLOT 0,191,255,0
70 END
```

La figura risultante e' quindi la seguente

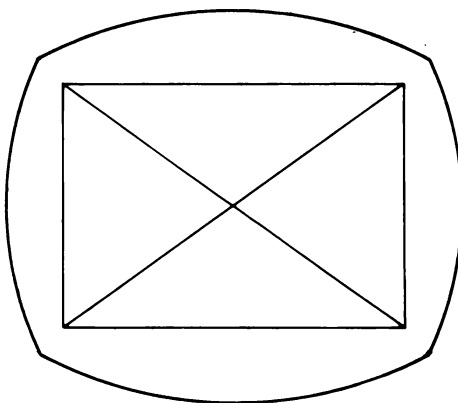



figura 33

Coloriamo un triangolo

Supponiamo di voler "colorare" uno dei 4 triangoli in cui e' diviso ora il rettangolo, ad esempio quello di destra. Non e' difficile. Basta ad esempio unire con dei segmenti i punti

```
da                 a

95,127 (centro)    255,0
95,127             255,1
95,127             255,2
 .................. e cosi' via fino a
95,127             255,191
```

Il programma necessario, da aggiungere al precedente, e' quindi

```
70 FOR I=0 TO 191
80 PLOT 127,95,255,I
80 NEXT
90 END
```

Se si preferisce tinteggiare a righine invece che in tinta unita basta ad esempio tracciare una riga si e una no. In questo caso e' sufficiente sostituire la linea 70 con

```
70 FOR I=1 TO 191 STEP 5
```

e verranno quindi tracciate dal centro le sole linee ai punti 255,0; 255,5; 255,10 ecc. La figura a pagina seguente illustra meglio quanto esposto.

Si osservi che alla fine della esecuzione il BASIC stampa i due punti (":") per indicare di essere in attesa di ordini.

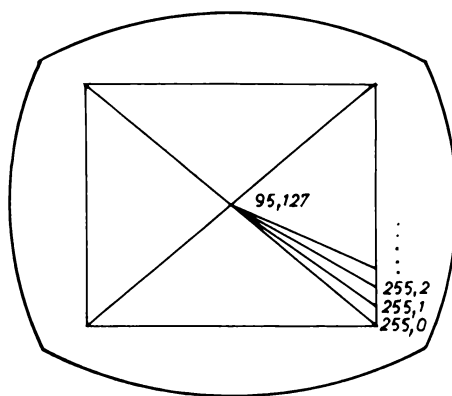



figura 34

Un programma curioso

Frequentemente nei giochi e' opportuno tracciare delle righe non esattamente regolari; ad esempio quando si desidera simulare il piano del terreno, quello del mare, i monti all'orizzonte. La frase PLOT disegna invece delle linee molto

regolari (almeno entro la risoluzione consentita dal video che come ricordiamo e' di 192x256 punti). Per ottenere lo scopo voluto e' tuttavia sufficiente l'uso di un programma molto semplice. Abbiamo gia' esaminato nei capitoli precedenti la funzione RND(0), destinata a calcolare un numero casuale compreso tra 0 ed 1. Proprio con essa raggiungeremo il nostro obbiettivo. Infatti tracciare una linea irregolare cosa significa?

1) Come ovvio scegliere un punto di partenza. Siano X0, Y0 le coordinate del punto di partenza. Ad esempio X0=0, Y0=170, che corrisponde a un punto circa in alto a sinistra.

2) Tracciare un piccolo segmento dal punto X0,Y0 ad un punto cosi' determinato:

> la x data da X0 piu' qualcosa; come dire che il punto di arrivo e' un po' piu' a destra. Piu' a destra di quanto? Proprio perche' vogliamo una linea irregolare, quel "qualcosa" lo scegliamo in modo casuale con la funzione RND(0) moltiplicata per 15 (ad esempio), in modo che i valori risultanti per questo qualcosa varino in modo casuale tra 0 e 15. Se la x del punto di arrivo la chiamiamo X1, scriveremo quindi
>
> X1 = X0 + RND(0) * 15
>
> la y data da Y0 piu' o meno qualcosa; il perche' del piu' o meno e' ovvio. Per avere la linea orizzontale irregolare sara' necessario che qualcuno dei punti di arrivo sia piu' alto di Y0 e qualcuno piu' basso. Per semplicita', vedremo dopo come, faremo in modo che i punti di arrivo calcolati siano alternativamente uno maggiore e l'altro minore di Y0. Quindi si avrebbe, con un ragionamento analogo a quello riportato per la X1 che
>
> Y1 = Y0 + RND(0) * 5
>
> dove il numero 5 potrebbe essere piu' grande per dare luogo ad irregolarita' piu' marcate o piu' piccolo per una linea meno frastagliata. Abbiamo detto pero' che

alcuni punti di arrivo devono essere piu' alti di Y0 per fare gli "zig" ed altri devono essere piu' bassi per fare gli "zag". Questo significa che la quantita' RND(0) * 5 deve essere una volta sommata ed una sottratta da Y0. Si scrive quindi

Y1 = Y0 + RND(0) * 5 * K

dove la variabile K vale una volta 1 ed una volta -1. Quando vale 1 il valore Y1 e' maggiore di Y0 ed il punto di arrivo e' piu' in alto. Quando vale -1 il valore di Y1 e' minore di Y0 e il punto di arrivo piu' basso.

3) <u>Tracciare il segmento</u> tra X0,Y0 e X1,Y1.

4) <u>Ripetere il procedimento</u> finche' la riga non e' lunga come si vuole ponendo di volta in volta il punto di partenza nel precedente punto di arrivo (facendo cioe' Y0=Y1 e X0=X1) e calcolando le nuove coordinate X1 ed Y1 per il nuovo punto di arrivo.

Ecco quindi infine il programma completo e, a pagina seguente, le figure che illustrano il procedimento.

```
10 REM PROGRAMMA PER TRACCIARE RIGHE IRREGOLARI
20 GRAPH
30 REM ASSEGNAZIONE DEL PUNTO DI PARTENZA
40 LEX X0=0
50 LET Y0=170
60 REM CALCOLO DEL PUNTO DI ARRIVO
70 LET K=1
80 LET X1=X0+RND(0)*15
90 LET Y1=Y0+RND(0)*5*K
95 REM K CAMBIA SEGNO PER LO ZIG-ZAG
100 LET K=K*-1
110 REM TRACCIAMENTO SEGMENTO
120 PLOT X0,Y0,X1,Y1
130 REM IL PUNTO DI ARRIVO DIVIENE DI PARTENZA
130 LET X0=X1
140 LET Y0=Y1
150 REM CONTROLLO LUNGHEZZA RIGA
160 IF X0<250 THEN GOTO 80
170 AT 23,0
180 END
```

Si osservino tre particolari:

1) La variabile K, che vale 1, viene moltiplicata per -1 ad ogni tracciamento di segmento. Pertanto vale la prima volta 1 (linea 70) e quindi -1 (dato dal prodotto di 1 per -1), poi 1 (dato dal prodotto di -1 per -1), quindi ancora -1 e cosi' via. Imprimetevi bene in mente questo meccanismo. Ne faremo ancora uso frequente.

2) Il punto di partenza (frasi 40, 50) e la lunghezza linea (frase 160) possono essere variate a piacimento. Linee verticali possono essere tracciate con ragionamenti simili scambiando le X e le Y.

3) La frase 170 e' stata aggiunta per evitare che il comparire dei due punti (":") alla fine del programma possa sciupare il nostro capolavoro.

Il lettore potra' provare a convertire in sottoprogramma il programma ora visto in modo da poterlo richiamare con una GOSUB passando il punto di partenza, la lunghezza e magari, in una variabile D, la direzione, orizzontale quando D=0 e verticale quando D=1.

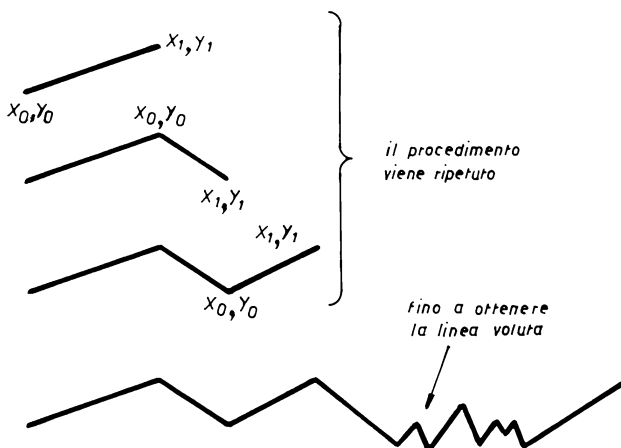
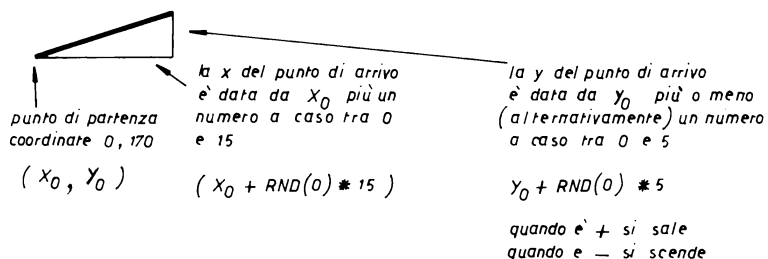



figura 35

Una osservazione

Da questo ultime esempio emerge, forse piu' che dai precedenti, che la stesura di un programma e' il frutto di due operazioni distinte:

1) l'elaborazione logica della procedura da seguire (in un italiano libero potrebbe essere definita "la pensata").

2) La traduzione della medesima in linee di programma.

algoritmo

E' chiaro che e' la prima ad essere la piu' importante; la programmazione e' solo la sua traduzione in pratica, esattamente come un buon disegno meccanico per una invenzione del pensiero. Per questo motivo ad essa e' stato riservato un nome particolarmente pomposo: "algoritmo". L'algoritmo e' svincolato da un particolare linguaggio di programmazione, proprio come l'invenzione meccanica e' svincolata da certi particolari di esecuzione: e' solo una lista di istruzioni elementari che un qualunque operatore, purche' dotato di una calcolatrice, puo' eseguire pedissequamente. L'algoritmo e' frutto di un processo detto di "analisi" del problema.

Un tempo era consuetudine diffusa che le applicazioni di un elaboratore venissero affidate a due persone distinte o classi di persone:

1) L'analista, ossia quello che studiava gli algoritmi.

2) Il programmatore, ossia quello che li traduceva in pratica convertendoli in programmi scritti in un certo linguaggio.

L'avvento dei piccoli computer ha tuttavia corretto questi ruoli riunendoli quasi sempre in una sola parsona (anche se lo sdoppiamento delle funzioni permane nei grandi centri di elaborazione); questo sia per un naturale processo di "smitizzazione" dell'elaboratore, oggi spesso programmato direttamente dalla persona cui e' affidato, che per la evoluzione dello stesso in direzione della comunicazione "umana": programmare in BASIC e' alla portata di tutti, programmare in assembler o in COBOL richiede studi ben precisi e talora una esatta conoscenza dell'hardware della macchina.

L'home computer ha accentuato questa concentrazione di mansioni; il propritario dell'apparecchio e' quasi sempre non solo l'analista ed il programmatore ma anche l'installatore, il tecnico di manutenzione, il personale di pulizia ecc. della sua macchina....

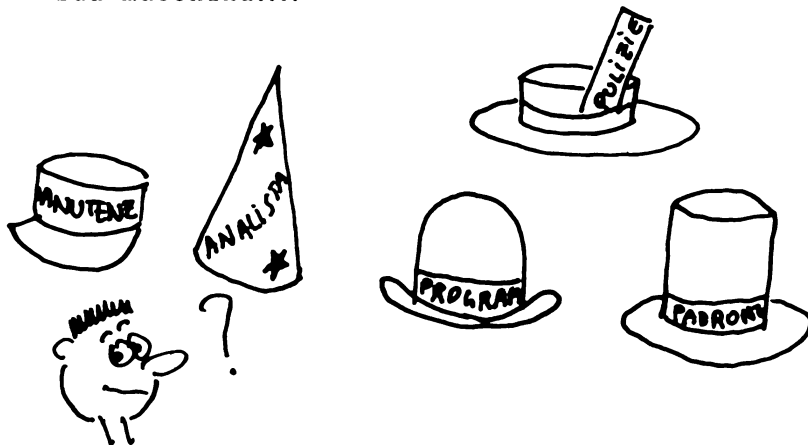



figura 36

Gli algoritmi vengono comunemente descritti per mezzo dei **diagrammi a blocchi** che fin qui non abbiamo ancora introdotto e che formeranno l'oggetto di un capitolo successivo. Importante per ora e' avere ben compreso la distinzione tra le fasi di analisi e di programmazione e **di aver capito che qualsiasi programma, prima di essere una lista di istruzioni, e' una sequenza logica descrivibile su carta per mezzo di parole comuni.**

BLANK/NOBLANK

Nel paragrafo precedente abbiamo parlato di concetti abbastanza "profondi". Ricreiamoci adesso con due istruzioni facili facili: la BLANK e la NOBLANK. Il loro uso e' semplicissimo:

BLANK oscura completamente il video, senza perderne i contenuti. Dopo la BLANK si puo' continuare a lavorare normalmente: lo schermo rimane pero' sempre buio.

NOBLANK ha l'effetto contrario: ripristina il display eventualmente aggiornato di tutte le modifiche aggiunte durante lo stato di BLANK.

Lo scopo di BLANK e NOBLANK e' quello di consen-

tire l'apparizione istantanea di un immagine preparata durante l'oscuramento. Un esempio di applicazione potrebbe essere quello di un programma destinato a misurare lo spirito di osservazione di un soggetto, facendo comparire un quadro e ponendo poi domande sul medesimo (un buon esercizio per casa!).

Il programma ARZPLOT

Ecco un nuovo esempio di un programma abbastanza divertente che genera sullo schermo un quadro astratto in modo casuale, il programma ARZPLOT (in un italiano pessimo: "PLOTtaggio ARZigogolato"). Questa volta procediamo al contrario; prima il programma e poi le osservazioni (il lettore diligente provera' a capirne il significato da solo prima di passare a leggere le spiegazioni; e' un ottima maniera per imparare).

```
10 REM PROGRAMMA ARZPLOT
20 GRAPH
30 REM PUNTO DI PARTENZA PER IL PRIMO TRATTO
40 LET X0=0
50 LET Y0=0
60 REM PUNTO DI ARRIVO PER IL PRIMO TRATTO
70 LET X1=100
80 LET Y1=100
90 REM PLOTTAGGIO
100 PLOT X0,Y0,X2,Y2
110 REM IL VECCHIO PUNTO DI ARRIVO DIVIENE PART.
120 LET X0=X1
130 LET Y0=Y1
140 REM CALCOLO CASUALE DEL NUOVO PUNTO DI ARR.
150 LET X1=RND(0)*255
160 LET Y1=RND(0)*191
170 REM E NUOVO CICLO
180 GOTO 100
190 END
```

La figura seguente mostra il risultato della esecuzione di ARZPLOT, esecuzione che continua fino a che non venga forzatamente interrotta.

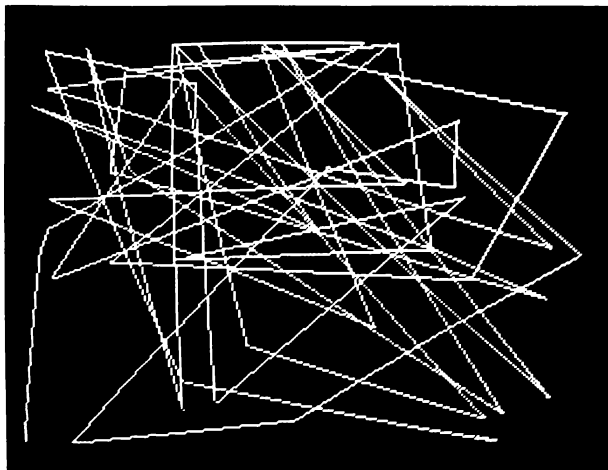



figura 37

L'algoritmo di questo programma e' semplicissimo. Descritto il primo tratto infatti si compiono solo le seguenti operazioni:

1) si prende come punto di partenza il punto di arrivo del tracciamento precedente;

2) si calcolano con la solita funzione generatrice di numeri casuali, due numeri (uno compreso tra 0 e 255 e uno tra 0 e 191) da usare come coordinate del nuovo punto di arrivo;

3) si esegue il tracciamento tra i due punti cosi' trovati;

4) si ricomincia ciclicamente dal passo 1.

Il programma e' di buon effetto e puo' servire da base per sperimentatori fantasiosi.

Un consiglio per gli autodidatti

Una buona maniera di apprendere le nuove tecniche di programmazione e' quella di studiare i programmi altrui, sia basandosi su testi appositi, come nel caso di questo volume, che per mezzo della rivista **cq elettronica.** Questo modo di procedere presenta numerosi vantaggi perche':

1) consente di imparare cose nuove;

2) esercita la mente alla comprensione di processi logici;

3) fa capire, meglio di qualsiasi discorso, come la documentazione non sia mai sufficiente.

Assi per diagrammi

Riprendiamo il nostro discorso sulla grafica illustrando con un esempio il modo di tracciare gli assi sopra lo schermo quando si debbano riportare dei diagrammi legati a quantita' numeriche, come il gia' citato diagramma dei profitti per periodo

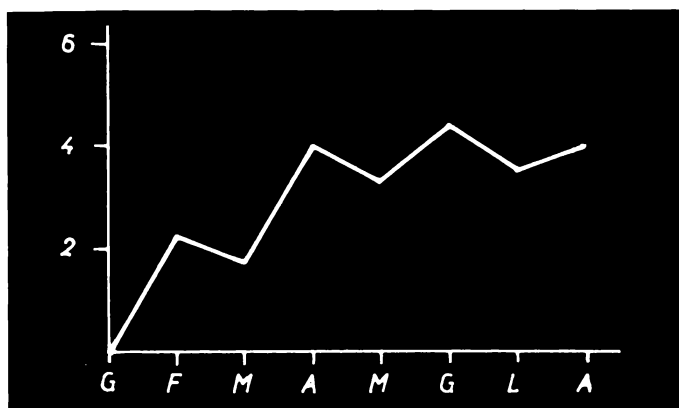



figura 38

Il problema consiste essenzialmente nel tracciare una riga munita di un certo numero di tacche in corrispondenza di ciascuna delle quali deve essere presente una didascalia, cifra o numero che sia (nell'esempio le lettere G, F, M, A, M, G, L, A, S, O, N, D, corrispondenti ai mesi Gennaio, Febbraio.....Dicembre ed i numeri 1, 2, 3... corrispondenti ai profitti espressi in milioni di dollari).

Per ottenere il risultato voluto nella forma migliore e' opportuno anche tenere presente le dimensioni fisiche dei caratteri, poiche', con l'istruzione AT che consente di posizionare eventuali scritte, possiamo solo spostarci di righe o di colonne intere e non di frazioni.

I caratteri standard sono sempre ricavati sia nel modo TEXT che nel GRAPH da una matrice di 6x8 punti, solo che, nel modo GRAPH, i caratteri sono

piu' spaziati tra di loro (infatti ce ne stanno 32 per riga invece di 40). Questo significa che nel modo grafico ogni carattere e' ricavato da una casella di 8x8 punti, con le due colonne di destra sempre oscurate e con le due righe in basso riservate ai caratteri discendenti tipo la "g" e la "p".

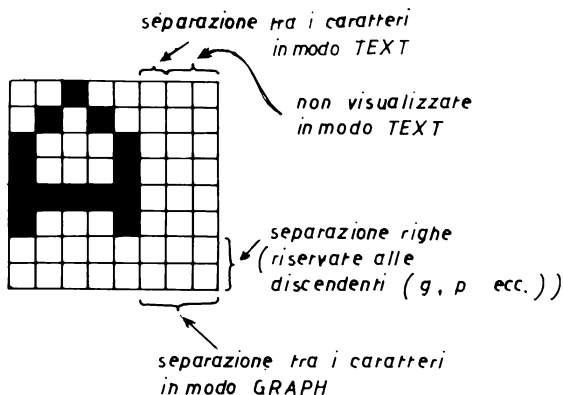



figura 39

Da quanto detto emerge che le scritte possono essere posizionate solo a partire da ascisse x multiple di 8 (cioe' a x=0, 7, 15 ecc.) e cosi' pure per le ordinate y (nota: questa limitazione sarebbe superabile; i mezzi per farlo sono pero' alla portata dei piu' esperti).

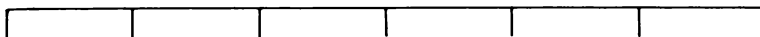
Iniziamo quindi la risoluzione del compito che ci siamo assegnati inventando un programma che descriva una linea di assegnata lunghezza munita ogni tanti punti di una tacca in corrispondenza della quale metteremo poi la relativa didascalia.

riga orizzontale con tacche

figura 40

Per semplicita' faremo in modo che il nostro programma tracci un numero N di segmenti ciascuno di lunghezza L terminante con una tacca rivolta verso il basso di 5 punti. La combinazione consecutiva di detti segmenti dara' luogo alla nostra linea calibrata.

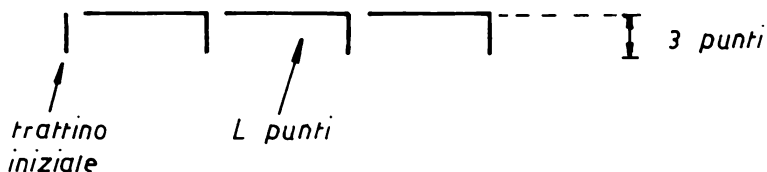



figura 41

Se il punto di partenza e' X0,Y0, un programma possibile e' il seguente

```
1000 REM PROGRAMMA ASSEX
1020 GRAPH
1022 REM - PRIMA TACCA
1023 PLOT X0,Y0,X0,Y0-3
1030 REM LA PARTE SEGUENTE E' RIPETUTA TANTE
1040 REM VOLTE QUANTE SONO I SEGM. RICHIESTI (N)
1050 FOR N1=1 TO N
1060 REM TRACCIAMENTO SEGMENTO
1070 REM LA FRASE SEGUENTE CALCOLA LA X DEL
1080 REM PUNTO DI ARRIVO COME SOMMA DI X0+L
1090 LET X1=X0+L
1100 REM LA Y DEL PUNTO DI ARRIVO E' SEMPRE Y0
1120 PLOT X0,Y0,X1,Y0
1130 REM TRACCIAMENTO DELLA TACCA VERTICALE
1140 PLOT X1,Y0,X1,Y0-3
1150 REM IL PUNTO DI PARTENZA PER IL PROSSIMO
1160 REM TRACCIAMENTO E' IL PUNTO DI ARRIVO
1170 REM DEL PRECEDENTE
1180 LET X0=X0+L
1190 REM LA Y0 RIMANE LA STESSA
1200 NEXT
1210 END
```

Sono convinto che i commenti inseriti molto abbondantemente nel programma (chiamiato ASSEX) rendano superflue altre spiegazioni. Il nostro ASSEX si presta particolarmente bene per essere usato come un sottoprogramma e richiamato per mezzo dell'istruzione GOSUB (questo e' il motivo per cui e' stato numerato a partire da 1000). Basta togliere la frase 1020 (se no tutte le volte che lo richiamiamo lo schermo si cancella) e sostituire la 1210 con un RETURN. Per effettuare il traccamento di un asse orizzontale bastera' assegnare il punto di partenza X0,Y0, la lunghezza L e eseguire una frase GOSUB 1000 (meglio se preceduta a un commento che ci ricorda la funzione del sottoprogramma

alla 1000). Riprendendo quindi l'esempio del diagramma dei profitti per periodo scrivendo un programma, che usi il sottoprogramma ASSEX che sta alla linea 1000, e che tracci una riga con 12 tacche, una per mese, dove poi apporremo le didascalie. Porremo una tacca ogni 16 punti (due caratteri). La lunghezza della linea sara' quindi di 11 segmenti x 16 punti = 176 punti. Poiche' lo schermo e' largo 256, ci avanzano 80 punti. Ne metteremo 50 a sinistra (dove andranno le didascalie dell'asse verticale) e 30 a destra. Lasceremo poi 4 righe sotto per eventuali scritte esplicative. Il punto di partenza del primo segmento e' quindi 50,32 (4 righe alte 8 punti). L, distanza tra le tacche, vale invece 16. Il programma principale diviene cosi':

```
410 REM PROGRAMMA DIAGRAMMA
420 GRAPH
430 REM PREDISPOSIZIONE PARAMETRI PER SOTT. ASSEX
440 LET X0=50
450 LET Y0=32
460 LET L=16
465 LET N=11
470 REM TRACCIAMENTO ASSE (SOTTOPROGRAMMA ASSEX)
480 GOSUB 1000
490 REM SCRITTURA DIDASCALIE
500 AT 21,6
510 PRINT "G F M A M G L A S O N D"
520 END
```

Abbiamo direttamente aggiunto anche, opportunamente posizionata, la scritta con le iniziali dei mesi.

L'asse verticale

Dal punto 50,32 partira' anche l'asse verticale, che sara' tracciato con un sottoprogramma del tutto simile ad ASSEX e che chiameremo ASSEY. Provi il lettore a scriverlo prima di verificarlo con quello sotto riportato.

```
2000 REM SOTTOPROGRAMMA ASSEY
2010 REM PER I COMMENTI VEDI ASSEX
2020 PLOT X0,Y0,X0-3,Y0
2030 FOR I=N1 TO N
2040 LET Y1=Y0+L
2050 PLOT X0,Y0,X0,Y1
```

```
2060 PLOT X0,Y1,X0-3,Y1
2070 LET Y0=Y0+L
2080 NEXT
2090 RETURN
```

Arricchiamo ora il programma DIAGRAMMA visto sopra in modo da completarlo con l'asse y (quello dei guadagni per periodo) aggiungendo le seguenti frasi

```
520 REM TRACCIAMENTO ASSE VERTICALE
530 LET X0=50
540 LET Y0=32
550 LET L=32
560 LET N=4
570 REM CHIAMATA AL SOTTOPROG. ASSEY
580 GOSUB 2000
590 AT 4,4
600 PRINT "4"
610 AT 8,4
620 PRINT "3"
630 AT 12,4
640 PRINT "2"
650 AT 16,4
660 PRINT "1"
670 AT 20,4
680 PRINT "0"
690 END
```

Come si puo' osservare sono state aggiunte anche le didascaie per l'asse verticale. Un tocco di perfezione puo' essere conferito con un po' di scritte ausiliarie:

```
690 AT 0,2
700 PRINT "Utili in Milioni di Dollari"
710 AT 23,14
720 PRINT "Mese"
730 END
```

Anche DIAGRAMMA come sottoprogramma

Il programma DIAGRAMMA da solo serve soltanto per disegnare un contorno, una struttura vuota, in cui dovremo poi inserire i dati di nostro interesse, ossia la curva che rappresenta il guadagno in ogni mese. E' da ritenere quindi che l'insieme di istruzioni del programma DIAGRAMMA debba essere richiamato dal programma che trac-

cia detta curva, ossia dal programma di calcolo propriamente detto. Meglio ancora, per ulteriore chiarezza, sara' creare un programma principale che richiami prima DIAGRAMMA e poi CALCOLO, essendo questo appunto quello di elaborazione dei dati, ed infine TRACCIA. Un programma cosi' fatto quindi

```
10 REM PROGRAMMA PRINCIPALE
20 REM TRACCIAMENTO ASSI E SCRITTE
30 GOSUB 400
40 REM CALCOLO GUADAGNI
50 GOSUB 3000
60 REM TRACCIAMENTO CURVA
70 GOSUB 4000
80 END
```

Come ovvio quanto sopra presuppone che DIAGRAMMA sia stato trasformato in sottoprogramma sostituendo la END con una RETURN e che siano stati scritti i sottoprogrammi 3000 e 4000, cioe' CALCOLO e TRACCIA, cosa questa che sara' oggetto dei paragrafi successivi.

Il programma TRACCIA

Abbiamo adesso l'occasione di fare un po' di esercizio con delle nostre vecchie conoscenze: le variabili multiple o vettori che abbiamo incontrato nella prima parte del volume. Vi ricordate l'esempio dell'incasso giornaliero? Supponiamo di avere invece una variabile U( ) che rappresenta l'utile di ogni mese in milioni di dollari. Avremo quindi che

U(1) e' l'utile di Gennaio
U(2) e' l'utile di Febbraio
U(3) e' l'utile di Marzo
....
U(12) e' l'utile di Dicembre

Per tradurre questi valori, che supponiamo per ora di avere gia' memorizzato (ci pensera' il programma CALCOLO), nel grafico che stiamo costruendo e' sufficiente il programma qui riportato:

```
4000 REM SOTTOPROGRAMMA TRACCIA
4040 FOR M=1 TO 11
4050 PLOT(M-1)*16+50,U(M)*32+32,M*16+50,
        U(M+1)*32+32
```

```
4060 NEXT
4070 RETURN
```

Nessuna paura per quella apparentemente complicata 4050! (che fra l'altro e' spezzata su due righe per esigenze tipografiche). E' tutto molto semplice. Infatti:

(M-1)*16+50 e' la x del punto da cui parte il tracciamento di ogni pezzetto "mensile" di curva. Infatti, quando M vale 1, la suddetta espressione vale X0, cioe' 50; per M=2 vale 56 e cosi' via, in corrispondenza quindi delle tacche mensili.

U(M)*32+32 e' la y del punto da cui parte il tracciamento, cioe' l'incasso mensile. Viene moltiplicata per 32 perche' sull'asse ogni tacca da una unita' corrisponde a 32 punti e ci si somma 32 perche' l'asse delle Y parte dal punto 50,32.

M*16+50 e' la x del punto di arrivo. Vale quanto la x del punto di partenza piu' 16. Tale e' la distanza tra due tacche.

U(M+1)*32+32 e' la y del punto di arrivo, cioe' e' il "guadagno del mese dopo". Proprio per questo la M varia da 1 ad 11 e non fino a 12.

In pratica quindi, grazie alla ripetitivita' dell'operazione data dalla frase FOR, viene tracciata una linea continua che unisce i punti corrispondenti ai guadagni mensili, la classica "spezzata", che ottimisticamente (e per semplicita') abbiamo previsto priva di punti ad ordinata negativa!

E il programma CALCOLO

E il programma CALCOLO? Si potrebbe rispondere ironicamente "se non lo sapete voi come calcolare i vostri guadagni..". Nell'ironia ci sarebbe pero' in fondo una certa logica, osservando a ritroso che in pratica il discorso fin qua e' rimasto molto sulle generali. Abbiamo visto assieme come costruire il diagramma con i relativi assi; poco importa se si parlava di guadagni "paperoniani" o delle temperature medie mensili o di qualsiasi altra relazione tra quantita' numeriche, come ad esempio il tracciamento di una funzione seno. Il programma CALCOLO e' quindi un compito

del lettore che lo realizzera' in funzione dei suoi obbiettivi. Se questi sono proprio dei ricavi espressi in milioni di dollari, ecco un semplice programma che consente di "veder funzionare" il tutto prima di passare ad altre piu' concrete applicazioni.

```
3000 REM SOTTOPROGRAMMA CALCOLO
3010 SERVE SOLO PER ASSEGNARE VALORI A I(1)..I(12)
3020 DIM U(12)
3030 FOR M=1 TO 12
3040 LET U(M)=RND(0)*4
3050 NEXT
3060 RETURN
```

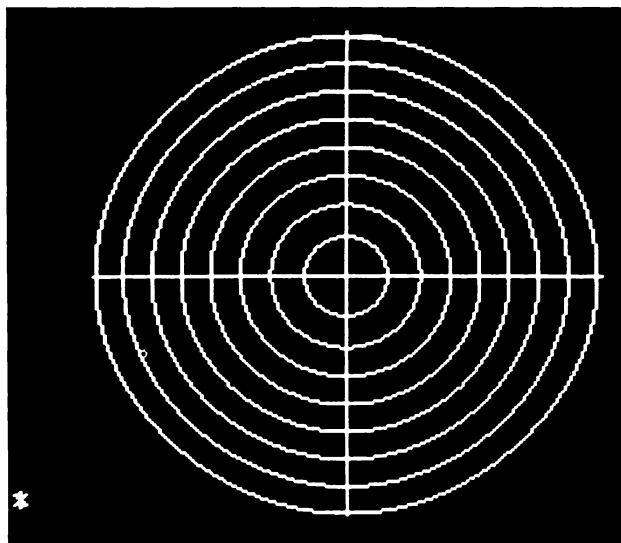
Si osservi che la presenza della frase DIM e' indispensabile ma che non deve essere molteplice. La frase DIM puo' cioe' comparire una sola volta, pena la segnalazione "ERRORE DI DIMENSION". Avrebbe potuto essere inserita anche nel programma principale.

```
1 GRAPH
2 ORG 127,95
3 GOSUB 1000
6 LET K=9025
10 FOR I=0 TO SQR(K/2)
20 LET P=SQR(K-I*I)
30 SET I,P
40 SET P,I
50 SET I,-P
60 SET P,-I
70 SET -I,P
80 SET -P,I
90 SET -I,-P
100 SET -P,-I
110 NEXT
115 LET A=-SQR(K)*20
116 LET K=K+A
117 IF K>0 THEN 10
130 GOTO 1050
1000 PLOT -95,0,96,0
1010 PLOT 0,-95,0,96
1020 RETURN
1050 AT 23,0
1060 END
```

figura 42
Alcuni cerchi concentrici generati con il semplice programma di cui sopra. Questa immagine può essere impiegata per la corretta taratura del monitor.

CAPITOLO X
Le animazioni

Le animazioni

Nel capitolo precedente abbiamo esaminato i rudimenti della grafica elementare descrivendo anche alcuni esempi. Si trattava pero' sempre di immagini "statiche", cioe' non in movimento. Quasi tutti gli home computers piu' evoluti consentono pero' tramite il BASIC, anche l'animazione, ossia la generazione di figure in movimento che, come si comprende, si prestano molto bene alla realizzazione di giochi.

figura 43

Quello della animazione e' forse l'aspetto dove maggiormente i vari BASIC differiscono. Anche in questo caso comunque queste disparita' non sono molto importanti: importante e' comprendere i concetti fondamentali che non sara' poi difficile trasferire sulle singole macchine.

Gli sprites

L'animazione, come la intenderemo noi, consiste essenzialmente in due aspetti fondamentali e nella loro combinazione:

1) nel movimento di una sagoma che trasla in vari punti dello schermo, ad esempio una palla che rimbalza contro le pareti;

2) nel susseguirsi di piu' sagome abbastanza simili nello stesso punto; ad esempio raffiguranti una ruota in varie posizioni di rotazione o un omino in varie posizioni mentre cammina. Il risultato sara' quello di vedere, come nei cartoni animati, la ruota in rotazione e l'omino che si muove al passo.

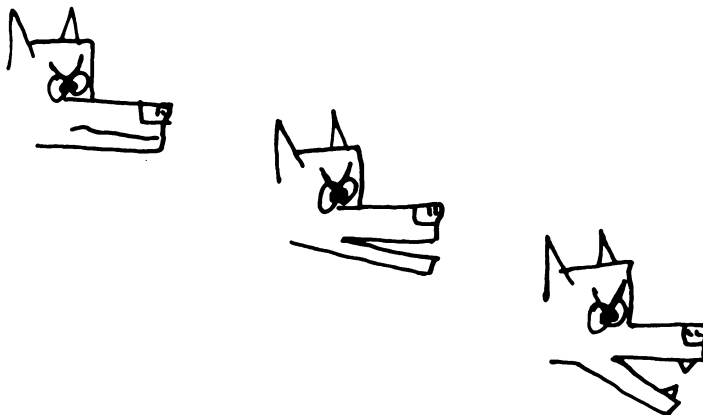



figura 44
L'animazione.

Queste "sagome" saranno d'ora in poi dette "sprites" (alla lettera "spiriti"), in accordo con la letteratura tecnica.

Vari tipi di sprites

Sul G5 possono essere generati 4 tipi di sprites, differenti per la costituzione e per le dimensioni. Caratteristica comune e' quella di essere ricavati da un reticolo di punti di forma quadrata.

Nel tipo 0-0 questo reticolo ha le dimensioni di 8x8 punti;

nel tipo 1-0 il reticolo e' di 16x16 punti, ma il

programmatore non puo' definirli tutti a proprio piacimento; si tratta puramente e semplicemente di un ingrandimento dello sprite di tipo 0-0 in scala doppia;

nel tipo 0-1 il reticolo e' di 16x16 punti e ogni punto puo' essere definito a piacimento;

nel tipo 1-1 il reticolo e' di 32x32 punti, ma il programmatore non puo' definirli tutti a proprio piacimento; si tratta puramente e semplicemente di un ingrandimento dello sprite di tipo 0-1 in scala doppia.

Il tipo dello sprite e' quindi definito da due numeri, il primo detto INGRANDIMENTO (si ha l'ingrandimento quando vale 1) ed il secondo TAGLIA (0= taglia piccola, 1=taglia grande).

Possono essere definiti fino a 32 sprites di taglia piccola o 8 di taglia grande ma essi non possono stare in piu' di 5 sulla stessa linea.

Uno sprite e' quindi di volta in volta ed a seconda delle circostanze un pallone, una nave spaziale, un meteorite, un carro armato, un giocatore di calcio, una bomba, un marziano ecc. ecc.

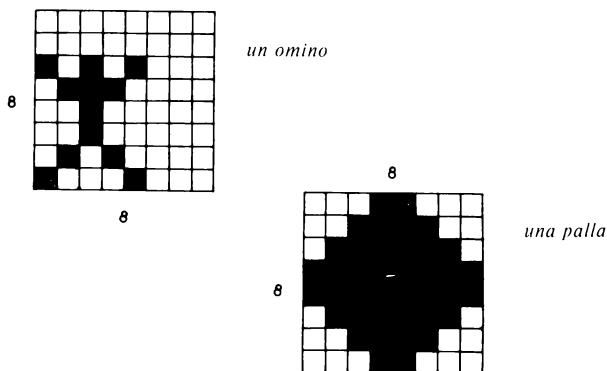



figura 45

Come usare gli sprites

Gli sprites sono numerati da 0 a 31. Quando due sprites si sovrappongono, lo sprite di numero inferiore "copre" lo sprite di numero piu' alto, producendo un effetto di prospettiva. Contemporaneamente agli sprites puo' essere usata la normale grafica (non dimentichiamo che stiamo operando nel modo GRAPH) che viene a trovarsi nell'ultimo piano prospettico (viene cioe' coperta

da qualunque sprite).

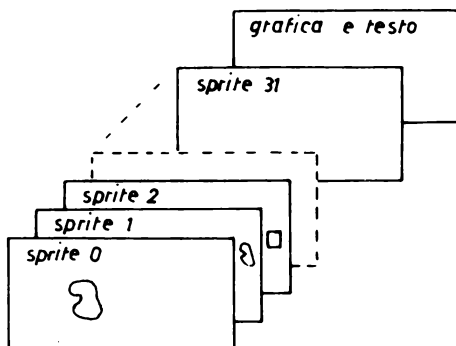



figura 46
Piani prospettici dell'immagine video.

Al comando GRAPH tutti gli sprite sono oscurati; questo, come ovvio, per evitare che compaiano, appunto come spiriti, quando si usa la normale grafica, e sono, per cosi' dire "disattivati". Prima di farne uso dobbiamo quindi attivarli per mezzo della istruzione SPTYP che si usa nella forma

SPTYP

SPTYP I,T,N

dove I e' una espressione il cui risultato finale sia 1 o 0 che indica l'INGRANDIMENTO;

T e' una espressione il cui risultato finale sia 1 o 0 che indica la TAGLIA;

N e' una espressione il cui risultato finale sia un numero compreso tra 0 e 31 che indica quanti sprite, a partire dal numero 0, desideriamo attivare.

Ad esempio:

SPTYP 0,0,1 attiva un solo sprite, il numero 0. Tutti gli sprite sono di tipo 0-0, cioe' di 8x8 punti.

SPTYP 1,0,4 attiva quattro sprite, dal numero 0 al numero 3. Tutti gli sprite sono di tipo 1-0, cioe'

di 16x16 punti ottenuti come ingrandimento doppio di uno sprite 8x8

SPTYP puo' essere usata come comando isolato o come istruzione di programma ma sempre in modo GRAPH.

Limiteremo la trattazione agli sprite di taglia piccola; quelli grandi sono descritti nel cap. XIII.

Disegnamo uno sprite

L'eseguire pero' l'istruzione SPTYP non da' luogo ad alcun risultato visibile; diciamo pure che gli sprites si trovano ora in stato di pre-attivazione. Questo e' dovuto a due fatti: il primo che gli sprite pur attivati non sono ancora disegnati, cioe' non gli abbiamo dato una forma. Il secondo che gli sprite si trovano fuori del campo dello schermo.

SPDEF

Disegnare uno sprite e' una cosa facilissima, grazie all'istruzione SPDEF che andiamo ad illustrare. Prima di pensare alla programmazione e' pero' necessario decidere la forma dello sprite, e cio' lo si fa benissimo con l'aiuto di una carta quadrettata, meglio se i quadretti sono grandi. Si traccia dapprima un contorno che racchiuda un quadrato di 8x8 caselle e quindi si segnano con la lettera O i punti destinati a restare bui e con la X quelli invece da illuminare (vedi figura a pagina seguente). Oltre a cio', ci sara' utile, numeriamo da 0 a 7 le otto righe orizzontali di otto quadretti che compongono la sagoma totale dello sprite. A questo punto il piu' e' fatto. Bastera' riportare di seguito alla SPDEF il numero dello sprite che stiamo definendo, il numero della riga e le X e le O dello schizzo con una F in fondo.

Esempio:

Disegnamo uno sprite approssimativamente a forma di palla nei limiti della risoluzione possibile.

come disegnare uno sprite

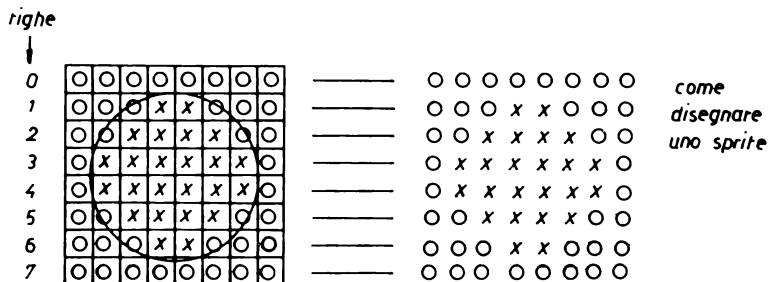



figura 47

Il sottoprogramma che lo definisce e' il seguente:

```
1000 REM SOTTOPROG. DEFINIZIONE PALLA
1010 SPDEF 0,0,OOOOOOOOF
1020 SPDEF 0,1,OOOXXOOOF
1030 SPDEF 0,2,OOXXXXOOF
1040 SPDEF 0,3,OXXXXXXOF
1050 SPDEF 0,4,OXXXXXXOF
1060 SPDEF 0,5,OOXXXXOOF
1070 SPDEF 0,6,OOOXXOOOF
1080 SPDEF 0,7,OOOOOOOOF
1090 RETURN
```

In parole la linea 1010 dice: "la riga 0 dello sprite 0 ha tutti i punti spenti; la linea 1 ha i primi tre punti spenti, due punti accesi e tre punti spenti; ecc.".

Nel definire uno sprite si puo' risparmiare un po' di fatica. Quando tutti i punti che seguono sono spenti si puo' infatti porre subito la lettera F senza tutti gli "OO..". Il sottoprogramma di cui sopra puo' quindi essere piu' rapidamente battuto cosi':

```
1000 REM SOTTOPROG. DEFINIZIONE PALLA
1020 SPDEF 0,1,OOOXXF
1030 SPDEF 0,2,OOXXXXF
1040 SPDEF 0,3,OXXXXXXF
1050 SPDEF 0,4,OXXXXXXF
1060 SPDEF 0,5,OOXXXXF
1070 SPDEF 0,6,OOOXXF
1090 RETURN
```

Si osservi che le linee 1010 e 1080 sono state omesse perche' e' inutile definire spente delle linee che lo sono gia' (questo non vale nel caso della ristrutturazione di uno sprite gia' definito).

Ogni linea dello sprite puo' essere definita indipendentemente dalle altre, o ridefinita, in qualsiasi ordine.

Come controllare posizione dello sprite

SPMOV

A questo punto il nostro sprite puo' prepararsi a fare la sua comparsa sullo schermo; basta solo che impariamo a controllarne la posizione. Anche questo e' facilissimo e ci sentiremo subito ricompensati della fatica fatta per la definizione. Basta fare uso della frase SPMOV seguita dal numero dello sprite e dalle coordinate in cui deve essere posizionato. Ad esempio:

```
SPMOV 0,100,128
```

posiziona lo sprite 0 nel punto di coordinate 100,128 (dopo la frase GRAPH gli sprite abilitati si trovano tutti nella posizione 0,0).

E' molto importante notare che quando parliamo delle coordinate di uno sprite si intenda parlare <u>delle coordinate dell'angolo in alto a sinistra del suo reticolo base</u> indipendentemente dal fatto che il punto che vi si trova sia acceso o spento. E' quindi ovvio che quando inizialmente lo sprite si trova nel punto 0,0 sia praticamente fuori dallo schermo.

Lo sprite puo' essere mosso quindi facilmente. Il ripetuto spostamento, come vedremo subito nell'esempio che segue, dara' luogo all'effetto del moto.

La palla che si muove

Facciamo ora muovere la palla, che abbiamo lasciato nella posizione 100,128, di moto uniforme alternato tra i punti 100,128 e 100,0. Sulla base delle cose sopra dette e' evidente che vedremo la palla quasi scomparire del tutto quando si trova all'estremo basso della corsa. Questo effetto e' prezioso perche' consente di far entrare in scena "gradualmente" gli sprite, come se si affacciassero davvero al bordo. Riepilogando un po' tutto quello che abbiamo detto, il programma diviene:

```
10 REM PROGRAMMA PALLA
20 GRAPH
30 REM UN SOLO SPRITE TIPO 0,0
40 SPTYP 0,0,1
45 SPCOL 0,15
50 REM DEFINIZIONE SPRITE 0 A FORMA DI PALLA
60 GOSUB 1000
70 REM K DA' LA VELOCITA'
80 LET K=1
84 LET Y=128
85 MOVIMENTO DELLO SPRITE
90 LET Y=Y-K
100 SPMOV 0,100,Y
105 REM CONTROLLO FINE CORSA
110 IF Y<1 THEN LET K=-K
120 IF Y>127 THEN LET K=-K
130 GOTO 90
140 END
```

A questo punto sono convinto che il lettore sia perfettamente in grado di seguire il programma: in pratica, sommando o sottraendo 1 alla coordinata verticale dello sprite si da' origine al moto del medesimo. Si osservi la tecnica usata per invertire K (e quindi la direzione del moto) al fine corsa. Se K fosse piu' grande, ad ogni SPMOV si avrebbe uno spostamento piu' sensibile della palla, accentuando l'effetto velocita'. Tuttavia K non puo' crescere oltre un certo limite, come il lettore vorra' sperimentare, o diverranno alla fine visibili dei veri "salti" tra una posizione e la successiva. Ricordo che i segni "<" e ">" stanno rispettivamente per "minore" e "maggiore".

Moti diversi

Nell'esempio precedente abbiamo determinato un moto alternato a velocita' costante. Questo tipo di movimento non e' tuttavia molto comune. Il pistone di una pompa ad esempio, si muove invece di moto uniformemente accelerato; il pistone parte da un estremo a velocita' nulla, quindi accelera per poi decelerare e raggiungere all'estremo opposto velocita' zero e cosi' via (forse puo' sembrare apparentemente strano che il pistone di una pompa resti fermo ma basta pensare che il moto si deve invertire ai punti morti per rendersi conto che la velocita' deve per forza, per fare cio',

passare per lo zero!).

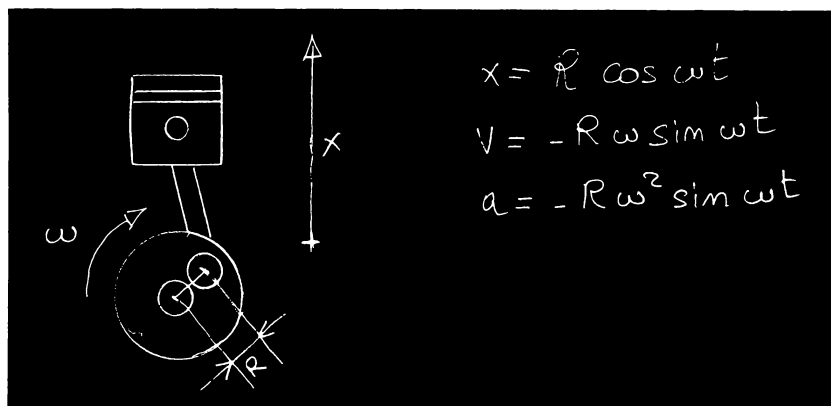



figura 48

Il moto del pistone puo' essere descritto circa da una funzione matematica di tipo sinusoidale. Non spaventatevi; questo vuol dire solo che le linee da 70 in giu' diventano

```
70 FOR I=1 TO 3.14159*2 STEP 0.1
80 SPMOV 0,100,SIN(I)*80+95
90 NEXT
100 GOTO 70
110 END
```

Il lettore potra' divertirsi a disegnare uno sprite a forma di pistone e un contorno dove farlo scorrere a forma di cilindro. Sia in questo esempio che nel precedente, per rendere piu' visibile l'oggetto in movimento basta procedere all'ingrandimento dello sprite modificando la linea 40 in

```
40 SPTYP 1,0,1
```

Altre spiegazioni

La perfetta comprensione dell'esempio precedente richiederebbe qualche nozione di fisica elementare, in particolare per la comprensione delle equazioni

$x = R \cos wt$ (1)
$v = - R w \sin wt$ (2)
$a = - R w^2 \sin wt$ (3)

dove w sta per la lettera greca "omega" per motivi

tipografici. Ma non facciamoci spaventare dai mulini a vento. In pratica le equazioni di cui sopra ci forniscono in forma molto semplice:

la 1: la posizione del pistone in funzione di R (la corsa totale del pistone), w, la velocita' di rotazione in giri al secondo dell'albero, ad un certo istante di tempo t.

la 2: con le stesse quantita', la velocita' ad un istante di tempo t.

la 3: sempre con le stesse convenzioni, l'accelerazione che subisce il pistone all'istante di tempo t.

Quindi chiunque, sulla base delle formule ora fornite, puo' calcolare la posizione, la velocita' lineare e l'accelerazione di un pistone sapendo la corsa (R) e la velocita' di rotazione dell'albero (w). "Ma io quei sin e cos non li ho mai calcolati!", dira' qualcuno. Ma il BASIC li sa calcolare benissimo. In BASIC le 1, 2 e 3 diventano

```
LET X = R * COS( W * T)                (1)
LET V = -R * W * SIN( W * T )          (2)
LET A = -R * W * W * COS( W * T)       (3)
```

(sono stati aggiunti un po' di spazi inutili per chiarezza). Vedete adesso come e' facile? Basta assegnare opportunamente i valori alle variabili per avere subito il risultato desiderato X, V od A.

Eccoci quindi al prossimo esercizio, che non riguarda direttamente gli sprite ma che prende origine dal precedente moto del pistone. Abbiamo detto sopra che la velocita' del pistone si annulla agli estremi della corsa. Vogliamo vederne, su un bel grafico, l'esatto andamento istante per istante? Niente di piu' facile. Basta questo semplice programma (il tracciamento degli assi di riferimento e' lasciato al lettore volenteroso):

```
10 REM PROGRAMMA TRACCIAMENTO VELOCITA' PISTONE
20 GRAPH
25 ORG 0,95
30 FOR T=0 TO 6.28 STEP 0.1
40 SET T*30,-80*SIN(T)
50 NEXT
```

```
60 END
```

Per semplicita' W e' stato fatto uguale ad 1 (per questo non compare) mentre la costante 30 che moltiplica T nella frase 40 e' stata aggiunta per "allungare" il grafico che sarebbe risultato se no troppo corto. R e' stato fatto uguale ad 80 in modo da avere una escursione complessiva di 160 punti sui 192 disponibili in verticale. Si osservi lo spostamento dell'origine ottenuto con la frase 25 in modo da avere l'asse x al centro dello schermo; si e' risparmiata cosi' qualche operazione aggiuntiva. Ecco alfine il risultato

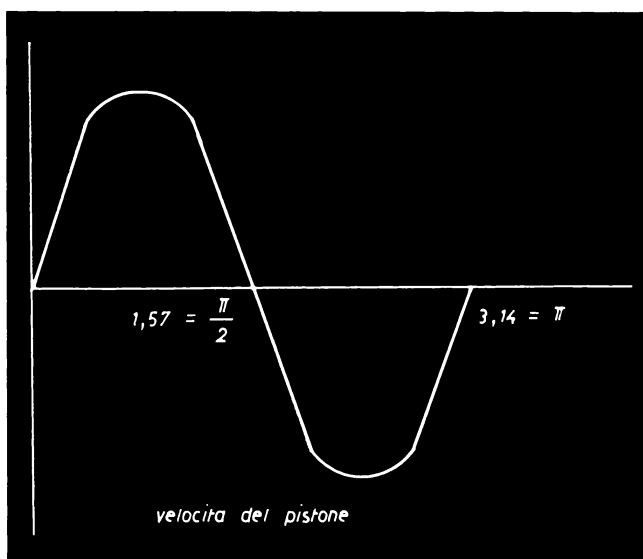



figura 49

Per chiarezza e' stato tracciato anche l'asse x. Vedete quindi? La velocita' parte da zero, aumenta (in senso negativo, il pistone scende) fino ad un massimo, poi continua a diminuire finche' diviene zero (attraversamento dell'asse x) e poi risale (questa volta in senso positivo, il pistone sale) finche' ancora decresce e cosi' via. Calibrando con opportune tacche gli assi si puo' cosi' conoscere in qualsiasi istante di tempo (cioe' in qualsiasi punto dell'asse x) quanto vale la velocita' lineare del pistone.

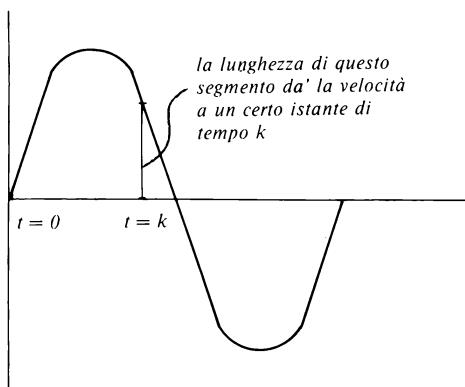



figura 50

Il lettore potra' scrivere dei programmi per tracciare, magari sullo stesso schermo, anche i diagrammi della posizione e della accelerazione e vedere cosi' come sono relati tra loro; si accorgera' che l'accelerazione e' massima quando la velocita' e' minima, cioe' agli estremi. Questo risultato e' in accordo con la logica: quando il pistone inverte il moto si ha infatti la massima accelerazione (pensate a starci sopra!).

La palla che rimbalza

Dopo una digressione di carattere "quasi scientifico" torniamo adesso ad argomenti piu' spassosi occupandoci della palla si che si muove sullo schermo e rimbalza quando urta contro i bordi del video. Chissa' quante volte, osservando un video gioco in un bar, avete pensato "chissa' come fa....". Apparentemente il problema sembra complesso e viene spontaneo pensare che dietro a cio' ci siano delle complicate equazioni di moto. In realta' la cosa e' di una semplicita' sbalorditiva.

Prima di pensare al rimbalzo vediamo come la palla puo' spostarsi in diagonale. Abbiamo gia' visto come la palla possa muoversi in verticale (esempi precedenti). Per farla muovere in diagonale basta agire anche sulla coordinata x dell'istruzione SPMOV. Infatti, supponendo che lo sprite si trovi alle coordinate 100,100, per determinarne un moto diagonale, bastera' incrementare non una ma entrambe le coordinate. Ad esempio

```
10 SPMOV 0,100,100
20 SPMOV 0,101,101
30 SPMOV 0,102,102
 .... ecc.
```

e' una maniera rozza ma intuitiva di far muovere lo sprite in diagonale (supponendo ovviamente che lo sprite sia gia' definito in tipo e forma). Un programma piu' completo e piu' logicc potrebbe essere il seguente:

```
10 REM PROGRAMMA DIAGONALE
20 GRAPH
30 SPTYP 0,0,1
40 REM DEFINIZIONE PALLA
50 GOSUB 1000
55 SPCOL 0,15
60 REM POSIZIONAMENTO INIZIALE
70 LET X=100
80 LET Y=60
90 FOR I=1 TO 100
100 SPMOV 0,X,Y
110 LET X=X+1
120 LET Y=Y+1
130 NEXT
140 END
```

(al solito il sottoprogramma 1000 non e' riportato). In questo modo lo sprite 0 viene spostato dalla posizione 100,60 per 100 volte in piccoli passi diagonali. Il programma PALLA visto precedentemente puo' similmente essere modificato cambiando la frase 100 in

```
100 SPMOV 0,Y,Y
```

cosicche' la palla si muova in diagonale e non in verticale.

Eccoci quindi all'"urto". Per sembrare reale, il rimbalzo deve essenzialmente sottostare ad una regola fondamentale: l'angolo di "uscita" deve essere uguale all'angolo di avvicinamento alla superficie ove si batte

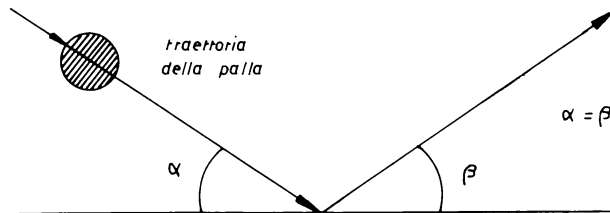



figura 51

Se noi muoviamo la palla modificando della stessa quantita' le coordinate dello sprite con l'istruzione SPMOV, il moto che risulta e' angolato di 45 gradi. Un moto inclinato a 45 gradi non e' altro che la combinazione simultanea, di due moti simili, uno verticale ed uno orizzontale.

Supponiamo nel programma sopra visto di cambiare la frase 70 con una

```
70 LET X=200
```

E' evidente che alla esecuzione la palla urtera' quasi subito contro il bordo destro dello schermo ma non si avra' certo il rimbalzo (non lo abbiamo ancora previsto) bensi' una segnalazione di errore, poiche' la coordinata X dello sprite diverra' presto maggiore del limite massimo ammesso di 255. E' quindi evidente che se vogliamo l'effetto del rimbalzo bisognera', dal momento in cui la coordinata X e' diventata 255, **non piu' incrementarla ad ogni passo bensi' decrementarla.** In questo modo il moto di traslazione verticale della palla continuera' (perche' continuiamo ad incrementare la Y), mentre quello orizzontale, decrementando la X, diverra' da destra verso sinistra (la coordinata X "va a calare"). C'e' solo un piccolo inconveniente: poiche' la SPMOV fissa le coordinate dell'angolo in alto a sinistra dello sprite, quando la X e' 255, lo sprite e' gia'

tutto fuori dal bordo destro. Bastera' quindi cambiare da incremento in decremento quando la X e' 255-8=247.

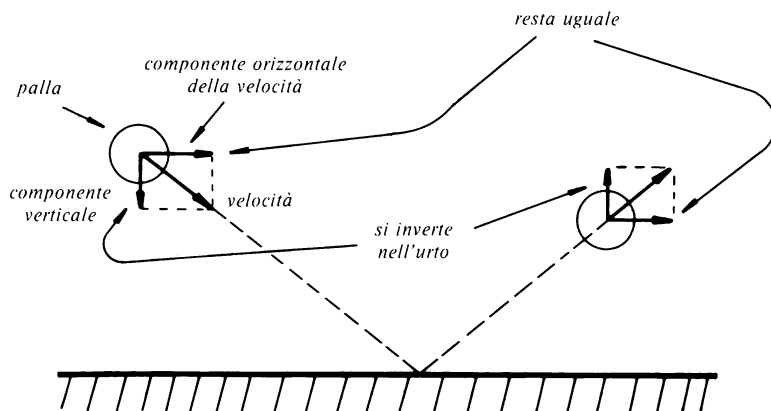



figura 52

La palla vagante

Traduciamo ora in pratica i concetti che abbiamo analizzato al punto precedente: desideriamo scrivere un programma che faccia muovere in continuita' la palla che urta e rimbalza sui quattro lati del video. Questo programma e' molto importante perche' costituisce la base di moltissimi giochi, sia di alcuni che vedremo in seguito sia di quelli che la vostra fantasia potra' suggerirvi. Vediamo subito il listato; faremo dopo i commenti:

```
10 REM PROGRAMMA RIMBALZO
20 GRAPH
30 SPTYP 0,0,1
40 REM PALLA
45 SPCOL 0,15
50 GOSUB 1000
60 REM PUNTO DI PARTENZA
70 LET X=20
80 LET Y=38
90 REM INCREMENTO PER IL MOVIMENTO
100 LET M1=1
110 LET M2=2
120 REM MOVIMENTO
130 LET X=X+M1
130 LET Y=Y+M2
140 REM CONTROLLO DI MARGINE
150 IF X>247 THEN GOTO 500
```

```
160 IF X<1 THEN GOTO 500
170 IF Y>191 THEN GOTO 600
180 IF Y<7 THEN GOTO 600
190 SPMOV 0,X,Y
200 GOTO 120
500 REM INVERSIONE MOVIMENTO ORIZZONTALE
510 LET M1=-M1
520 GOTO 120
600 REM INVERSIONE MOVIMENTO VERTICALE
610 LET M2=-M2
620 GOTO 120
```

Fino alla 70 e' la solita sequenza. La 70 e la 80 fissano un punto di partenza scelto a caso. La 100 e la 110 prefissano nelle variabili M1 ed M2 le quantita' che verranno via via sommate alle coordinate dello sprite per dare il movimento. Piu' sono grandi, maggiore sara' la velocita' (con il solito limite al quale si comincia a vedere il "salto"). Il movimento viene ottenuto semplicemente sommando M1 ed M2 alle coordinate ma prima della SPMOV (frase 190) si esegue il controllo di margine, verificando se le coordinate ottenute rientrano nello schermo. Diversamente (500 e 600) si provvede ad invertire il movimento che tende a portare fuori campo.

Al comando RUN la palla iniziera' a rimbalzare tra le pareti del video con un effetto molto piacevole.

La funzione SPCOIN(0)

Non appena realizzato il programma RIMBALZO proverete subito il desiderio di renderlo un po' piu' animato aggiungendo la vostra partecipazione alle evoluzioni della palla. Il primo istinto sara' quello di aggiungere un'altro sprite, ad esempio a forma di racchetta, cosi' da poter giocare al ping-pong. Sorge pero' subito un problema: come sapere quando due sprite si toccano? Sarebbe possibile fare dei paragoni sulle coordinate ma il metodo sarebbe un po' lungo e complicato. Abbiamo invece a disposizione una apposita funzione che soddisfa questa necessita': la SPCOIN(0) (COINcidenza di SPrite). Infatti la funzione SPCOIN(0) puo' dare solo due valori:

0 quando sul video non ci sono sprite che si

toccano

1 quando sul video ci sono almeno due sprite che si toccano

La SPCOIN(0) puo' essere usata esattamente come qualsiasi altra funzione; ad esempio

PRINT SPCOIN(0)

oppure

IF SPCOIN(0)=1 THEN PRINT "SCONTRO!!!"

Si osservi che la SPCOIN(0) e' una funzione "intelligente"; infatti per determinare il contatto tra due sprite non si limita a controllare le coordinate ma guarda proprio i punti illuminati. Si ha cosi' il valore 1 in uscita non quando i reticoli base sono in contatto ma quando lo sono i punti che formano lo sprite visibile (cosa utilissima nel caso di sprite frastagliati).

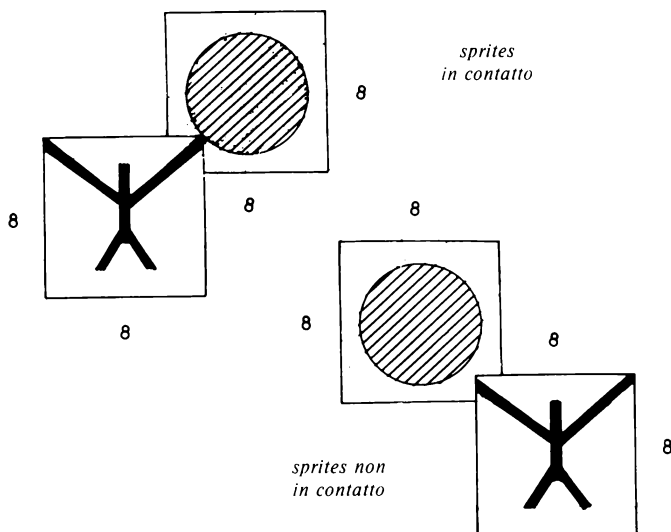



figura 53

Il colore degli sprites

SPCOL

Gli sprites possono essere di qualsiasi dei 16 colori ammessi (in bianco e nero diventano otto tonalita'). Il colore dello sprite puo' essere modificato con l'istruzione SPCOL. Basta scrivere

SPCOL 2,10

perche' lo sprite 2 assuma il colore 10, cosi' come definito nella sottostante tabella:

Codice colore	colore	(in b & n)
0	TRASPARENTE	TRASPARENTE
1	NERO	NERO
2	VERDE MEDIO	TONO 1
3	VERDE CHIARO	
4	BLU SCURO	TONO 2
5	BLU CHIARO	
6	ROSSO SCURO	TONO 3
7	CYAN	
8	ROSSO MEDIO	TONO 4
9	ROSSO CHIARO	
10	GIALLO SCURO	TONO 5
11	GIALLO CHIARO	
12	VERDE SCURO	TONO 6
13	MAGENTA	
14	GRIGIO	TONO 7
15	BIANCO	

Quando due sprites di colore diverso hanno dei punti in comune l'effetto prospettico della sovrapposizione e' ancora piu' marcato.

Un gioco completo il pallone

Ecco che adesso abbiamo gli elementi per costruire il primo gioco completo e divertente; dopo averlo realizzato ci passerete certamente diverse ore ed i vostri familiari vorranno misurare con voi la loro abilita'. Il gioco consiste nel muovere per mezzo di alcuni tasti un portiere, cioe' uno sprite a forma di omino con le braccia rivolte verso l'alto, in modo da "parare" un palla che rimbalza contro le pareti laterali del video e contro una parete, sul bordo superiore, che si "abbassa" ad ogni rete subita. Dei messaggi che compaiono in cima allo schermo, informano del numero dell reti subite e di quelle parate. Piu' reti si subiscono piu' il soffitto "scende" e piu' difficile quindi diventa il gioco. La partita termina quando il giocatore ha compiuto 21 parate e quindi vince o ha subito 21 reti e quindi perde.

Il portiere viene spostato per mezzo di tre tasti, l' "1", il "2" ed il "3". L' "1" muove il portiere

verso sinistra (ed il portiere resta in moto fino al bordo), il "2" lo ferma nella posizione in cui si trova ed il "3" funziona come il "2" ma verso destra.

figura 54

Per aumentare l'effetto di realta' del gioco, il portiere nelle parate compie un piccolo salto, migliorando la spettacolarita'. Grazie al modo di operazione della funzione SPCOIN(0), che si basa sulla reale coincidenza dei punti illuminati degli sprites, possono inoltre verificarsi delle parate piu' complesse, anche in tre "tempi" o addirittura delle autoreti.

Sulla base delle informazioni che abbiamo precedentemente fornito il programma non presenta nessuna difficolta' di comprensione al di fuori dell'uso della funzione INP(244) che serve a "leggere" i tasti che muovono il portiere e che e' dettagliatamente descritto nel paragrafo seguente.

Il programma si presta a molte modifiche e puo' servire da base per giuochi completamente differenti. Suggerisco al lettore, una volta realizzatolo e collaudatolo nella forma base, di tentare qualche "personalizzazione"; potrebbe trattarsi di qualcosa di molto semplice, come un migliore disegno del campo da calcio, a qualcosa di piu' complesso, come il trasformare il programma per due giocatori.

```
1 REM PROGRAMMA PALLONE
2 GRAPH
3 PLOT 0,0,255,0
5 GOSUB 1000
20 SPCOL 0,11
30 SPCOL 1,15
```

```
40 SPTYP 1,0,2
70 LET P1=20
80 LET P2=38
90 LET M1=4
100 LET M2=M1
105 LET R1=120
106 LET S=0
107 LET T=191
108 LET P=0
110 LET P1=P1+M1
120 LET P2=P2+M2
130 IF P1>247 THEN GOTO 500
140 IF P1<1 THEN GOTO 500
150 IF P2<1 THEN GOTO 700
160 IF P2>T THEN 600
170 SPMOV 0,P1,P2
171 IF SPCOIN(0)=1 THEN GOTO 800
172 IF INP(255)=206 THEN LET R1=R1-4
173 IF INP(255)=204 THEN LET R1=R1+4
174 SPMOV 1,R1,18
180 GOTO 110
500 LET M1=M1*-1
510 GOTO 110
600 LET M2=M2*-1
610 GOTO 110
700 AT 0,0
710 LET S=S+1
730 UNPLOT 0,T,255,T
735 PRINT "RETI SUBITE ",S
736 IF S=19 THEN GOTO 1500
740 LET T=T-10
745 PLOT 0,T,255,T
750 GOTO 600
800 LET P2=P2+10
810 SPMOV 1,R1,25
820 LET P=P+1
821 IF P=21 THEN 1600
830 AT 0,0
835 FOR I=1 TO 100
836 NEXT
840 PRINT "RETI PARATE ", P
850 SPMOV 1,R1,18
860 GOTO 600
1000 SPDEF 0,0,OF
1010 SPDEF 0,1,OOOXXF
1020 SPDEF 0,2,OOXXXXF
1030 SPDEF 0,3,OXXXXXXF
```

(continua a pagina seguente)

```
1040 SPDEF 0,4,OXXXXXXF
1050 SPDEF 0,5,OOXXXXF
1060 SPDEF 0,6,OOOXXF
1070 SPDEF 0,7,OF
1080 SPDEF 1,0,XOOXOOXXF
1090 SPDEF 1,1,OXOXOXOOF
1100 SPDEF 1,2,OOXXXOOOF
1110 SPDEF 1,3,OOOXOOOOF
1120 SPDEF 1,4,OOOXOOOOF
1130 SPDEF 1,5,OOXOXOOOF
1140 SPDEF 1,6,OXOOOXOOF
1150 SPDEF 1,7,XOOOOOXOF
1160 RETURN
1500 AT 10,0
1510 PRINT "TATTAMEA..."
1520 PRINT
1530 PRINT "T A T T A M E A ....!"
1540 GOTO 1630
1600 AT 10,0
1610 PRINT "BRAVO!"
1615 PRINT
1620 PRINT "B R A V O!"
1630 END
```

Comandi istantanei

La frase INPUT e' una frase che serve per l'ingresso dei dati in un programma. E' pero' una frase di ingresso evoluto ma lento. Evoluto, perche' capace di presentare un messaggio e di convertire numeri anche complessi nella rappresentazione interna della macchina; lento perche' richiede sempre l'immissione del ritorno carrello a fine introduzione del dato. Per muovere il portiere noi abbiamo bisogno invece di maggior rapidita' di azione. Vogliamo cioe' che quando si preme un tasto, il suo effetto venga subito rilevato "in continuazione" senza che sia necessario premere il RETURN. A questo scopo provvede la funzione INP(244) che "fotografa" la situazione della tastiera fornendo indietro un valore corrispondente al codice del tasto premuto (la tabella dei codici e' riportata nelle pagine che seguono). Ci si rende facilmente conto di come operi la INP(244) in questo semplice programma

```
10 PRINT INP(244)
20 GOTO 10
```

Quando nessun tasto e' premuto, il codice ritorna-

to dalla INP(244) e' quello dell'ultimo tasto battuto. E' ovvio che questa funzione e' preziosa in moltissimi programmi perche' consente di accrescere l'interattivita' tra l'operatore e la macchina.

Guerra di carri armati

Passiamo a descrivere brevemente adesso un altro programma che usa gli sprites: la guerra dei carri armati. Il gioco consiste nello sparare con un carro armato mobile lungo l'asse y contro un muro evitando di essere colpiti da parte di due bombe che attraversano lo schermo in modo casuale. Dei tasti consentono di spostare il carro armato, come il portiere nel programma PALLONE mentre un tasto in piu' serve per sparare. I colpi sono limitati in numero.

Animazioni

Abbiamo detto che possiamo realizzare delle animazioni. L'animazione funziona sullo stesso identico principio del cartone animato (e del cinematografo in generale), ossia sulla rapida successione di immagini di poco dissimili che danno l'impressione della continuita' nel movimento. Chiaramente un home computer non ha una definizione tale da poter generare cartoni animati di tipo disneyano ma consente ugualmente di ottenere nei giochi un effetto soddisfacente.

Per introdurci all'animazione ci baseremo sull'esempio precedente; il nostro obiettivo e' quello di fare in modo che quando il carro armato e' colpito dalla bomba si abbia l'impressione dell'esplosione. Questo significa in pratica sostituire al carro, magari alternandola un paio di volte, uno sprite a forma di "nuvoletta" e quindi nel far sparire completamente entrambi.

SPFORM

Il G5, come altri home computers, e' provvisto di una apposita funzione per questo scopo; la SPFORM. Grazie ad essa e' possibile assegnare la forma di uno sprite ad un altro sprite, o, in altre parole, scegliere per uno sprite una forma gia' definita per un altro sprite.

Ad esempio: supponiamo di avere definito per lo sprite 0 la forma del carro armato e per lo sprite 1 la forma della nuvoletta. Con l'istruzione

```
SPFORM 0,1
```

assegnamo allo sprite 0 la forma della nuvoletta. Da quel momento lo sprite 0 avra' sempre la forma di nuvoletta, almeno finche' non decideremo di fargli assumere ancora la forma del carro con una frase

```
SPFORM 0,0
```

Verrebbe quindi spontaneo di risolvere il problema assegnato (lo scoppio del carro armato), con una serie di

```
 .. SPFORM 0,1
 .. SPFORM 0,0
 .. SPFORM 0,1
 ..
```

ma c'e' un piccolo inconveniente; in questo modo la succesione delle forme diverse e' troppo rapida, almeno per questo scopo specifico, mentre la giusta tempificazione e' molto importante ai fini dell'effetto. Nessun problema quando si tratta di perdere tempo. Basta inserire tra le SPFORM una chiamata ad un sottoprogramma "inutile" il cui scopo sia solo quello di perdere tempo, come ad esempio

```
2000 FOR J1=1 TO 1000
2010 NEXT
2030 RETURN
```

Si deve porre attenzione affinche' la variabile utilizzata nel ciclo del sottoprogramma di attesa non sia usata anche nel programma principale o avremo la brutta sorpresa di ritrovarla a valore 1000!

Nella lista precedente del programma CARRO sono riportate anche le istruzioni necessarie per completarlo con l'effetto di scoppio, ottenuto con due sprite "nuvoletta".

```
1 REM GUERRA DI CARRI ARMATI
10 GRAPH
15 LET P1=255
16 LET C=50
20 REM DEFINIZIONE SPRITE
30 GOSUB 1000
50 PLOT 245,0,245,185
75 LET P1=255
80 LET P2=INT(RND(0)*191)
85 LET P3=INT(RND(0)*191)
86 LET O=50-INT(RND(0)*40)
90 LET P1=P1-10
95 IF P1<1 THEN 75
100 SPMOV 0,P1,P2
110 SPMOV 2,P1,P3
150 IF INP(255)=206 THEN GOTO 5000
160 IF INP(255)=200 THEN GOTO 5100
162 IF R1>191 THEN LET R1=191
164 IF F=1 THEN GOTO 180
165 IF INP(255)=202 THEN GOSUB 2000
180 SPMOV 1,25,R1
185 IF SPCOIN(0)=1 THEN GOTO 3000
190 GOTO 90
1000 SPCOL 0,11
1010 SPCOL 1,15
1015 SPCOL 2,10
1020 SPTYP 1,0,3
1025 SPFORM 2,0
1030 SPDEF 1,0,OF
1040 SPDEF 1,1,OF
1050 SPDEF 1,2,OF
1060 SPDEF 1,3,XXXXXXXOF
1070 SPDEF 1,4,OXOOXXOOF
1080 SPDEF 1,5,OXOOOOXXF
1085 SPDEF 1,6,OXOOXXOOF
1090 SPDEF 1,7,XXXXXXXOF
1100 SPDEF 0,0,OF
1110 SPDEF 0,1,OOOOXXOOF
1120 SPDEF 0,2,OXXXXOOOF
1130 SPDEF 0,3,XXXXXOOOF
1140 SPDEF 0,4,OXXXXOOOF
1150 SPDEF 0,5,OOOOXXOOF
1160 SPDEF 0,6,OF
1170 SPDEF 0,7,OF
1180 SPDEF 3,0,OOOXOOXF
1190 SPDEF 3,1,OXOXOOXOF
1200 SPDEF 3,2,OOXOXXOXF
1210 SPDEF 3,3,XOXXOXXXF
1220 SPDEF 3,4,OXXXXXOOF
1230 SPDEF 3,5,OXOXOXOOF
1240 SPDEF 3,6,XOOXOXOOF
1250 SPDEF 3,7,XOOOOOXOF
1260 RETURN
2000 UNPLOT 245,R1-15,245,R1-7
2030 LET C=C-1
2040 AT 0,0
2050 PRINT "COLPI RESTANTI ",C
```

```
2060 IF C=0 THEN GOTO 3000
2070 LET F=1
2080 RETURN
3000 SPFORM 1,3
3001 SPCOL 2,0
3002 SPCOL 0,0
3010 GOSUB 4000
3020 SPFORM 1,1
3030 GOSUB 4000
3031 SPFORM 1,3
3032 GOSUB 4000
3033 SPFORM 1,1
3034 GOSUB 4000
3035 SPFORM 1,3
3036 GOSUB 4000
3037 SPFORM 1,1
3038 GOSUB 4000
3040 SPFORM 1,3
3041 GOSUB 4000
3042 SPCOL 1,0
3050 AT 0,0
3060 PRINT "C O L P I T O ! ! !"
3070 GOTO 4030
4000 FOR I=1 TO 20
4010 NEXT I
4020 RETURN
4030 END
5000 LET R1=R1-6
5010 GOTO 5110
5100 LET R1=R1+6
5110 LET F=0
5120 GOTO 162
```

Altre applicazioni di SPFORM

Ma la SPFORM ha anche altre applicazioni come vedremo nell'esempio che segue. Supponiamo di voler realizzare un programma per simulare il gioco dei dadi.

Dopo avere disegnato le sagome dei cubi con l'uso della frase PLOT, vogliamo riportare sulla faccia anteriore i punti come in un vero dado e poi farli cambiare a piacere. Basta definire 6 sprite ciascuno a forma della faccia di un dado e quindi, con l'istruzione SPFORM, assegnare agli sprite 0 ed 1, preventivamente posizionati al centro della faccia desiderata, la forma del punto che si vuole ottenere (magari ottenuto casualmente con la funzione RND(0).

Facendo l'esempio con un solo dado e trascurando il disegno della sagoma esterna, si ha

```
10 REM PROGRAMMA DADO
15 GRAPH
20 REM DEFINIZIONE 6 SPRITE PER FORMA FACCIE
30 GOSUB 1000
40 SPMOV 0,100,100
50 SPFORM 0,RND(0)*5
60 END
```

Si osservi che gli sprite sono numerati da 0 a 5 e che corrispondono ai punti da 1 a 6. Desiderando eseguire anche una totalizzazione del punteggio si preferira' far cosi':

```
50 LET P=RND(0)*5
60 SPFORM 0,P
70 LET T=T+P+1
80 END
```

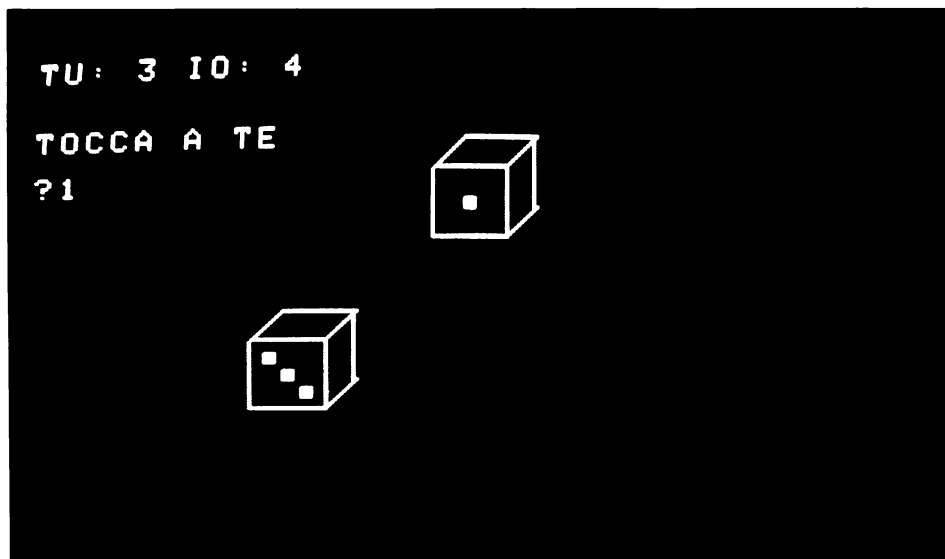
dove in T si accumula il totale dei punti P piu' uno, perche' P va da 0 a 5 invece che da 1 a 6 (questo ovviamente nel caso che il programma prosegua ciclicamente; la frase 70 puo' essere espressa in parole dicendo: poni il totale T uguale al totale fin qui calcolato T piu' i punti P piu' 1).

Si riporta in chiusura la lista di un possibile programma DADI che il lettore vorra' studiare e migliorare; come per gli esempi precedenti non si e' infatti proceduto ad una ottimizzazione delle procedure riportate allo scopo di lasciare al lettore questo interessante compito.

```
1 REM PROGRAMMA DADI
2 GRAPH
3 SPTYP 1,0,7
6 SPMOV 1,100,100
10 SPDEF 1,3,OOOXXF
20 SPDEF 1,4,OOOXXF
30 SPDEF 2,1,OXXF
40 SPDEF 2,2,OXXF
50 SPDEF 2,5,OOOOOXXF
60 SPDEF 2,6,OOOOOXXF
70 SPDEF 3,0,XXF
80 SPDEF 3,1,XXF
90 SPDEF 3,3,OOOXXF
100 SPDEF 3,4,OOOXXF
```

```
110 SPDEF 3,6,OOOOOOXXF
120 SPDEF 3,7,OOOOOOXXF
130 SPDEF 4,1,OXXOOXXF
140 SPDEF 4,2,OXXOOXXF
150 SPDEF 4,5,OXXOOXXF
160 SPDEF 5,0,XXOOOOXXF
170 SPDEF 5,1,XXOOOOXXF
180 SPDEF 5,3,OOOXXF
190 SPDEF 5,4,OOOXXF
200 SPDEF 5,6,XXOOOOXXF
210 SPDEF 5,7,XXOOOOXXF
220 SPDEF 6,0,XXOXXOXXF
230 SPDEF 6,1,XXOXXOXXF
240 SPDEF 6,3,XXOXXOXXF
250 SPDEF 6,4,XXOXXOXXF
260 SPDEF 6,6,XXOXXOXXF
270 SPDEF 6,7,XXOXXOXXF
280 LET E1=70
285 LET E2=94
290 LET P=10
300 GOSUB 1000
310 LET E1=130
320 LET E2=154
330 GOSUB 1000
340 SPMOV 1,74,90
350 SPMOV 2,134,150
380 SPCOL 1,15
390 SPCOL 2,15
400 LET T=0
410 LET C=0
420 AT 3,0
430 PRINT "TOCCA A TE"
435 INPUT A
440 GOSUB 2000
450 LET T=T+D1+D2
460 AT 0,0
470 PRINT "TU:",T,"IO:",C
471 FOR I=1 TO 1000
472 NEXT
480 AT 3,0
490 PRINT "TOCCA A ME"
500 GOSUB 2000
510 LET C=C+D1+D2
514 AT 0,0
515 PRINT "TU:",T,"IO:",C
520 IF C>21 THEN GOTO 800
530 IF T>21 THEN GOTO 900
540 FOR I=1 TO 1000
550 NEXT
560 GOTO 420
800 AT 2,0
805 IF T>C THEN GOTO 900
806 IF C=T THEN GOTO 950
810 PRINT "HO VINTO IO, TATTAMEA!"
820 GOTO 3000
```

```
900 AT 2,0
910 PRINT "GRRR.. HAI VINTO TU!"
920 GOTO 3000
950 AT 2,0 PRINT "PARI"
960 GOTO 3000
1000 REM SOTTOPROGRAMMA PER DISEGNARE CUBI
1010 REM E1 PUNTO DI PARTENZA BASE - E2 ARRIVO
1020 LET E3=E2+P
1030 LET E4=E1+P
1200 PLOT E1,E1,E2,E1
1210 PLOT E2,E1,E2,E2
1230 PLOT E2,E2,E1,E2
1240 PLOT E1,E2,E1,E1
1250 PLOT E2,E2,E3,E3
1260 PLOT E1,E2,E4,E3
1270 PLOT E2,E1,E3,E4
1280 PLOT E4,E3,E3,E3
1290 PLOT E3,E3,E3,E4
1300 RETURN
1310 END
2000 LET D1=INT(RND(0)*5)+1
2010 LET D2=INT(RND(0)*5)+1
2020 SPFORM 1,D1
2030 SPFORM 2,D2
2040 RETURN
3000 END
```

figura 55

CAPITOLO XI
Le funzioni

Le funzioni

In BASIC una funzione non e' altro che una "scatoletta" in cui entra un numero ed esce un altro numero. Ad esempio, abbiamo gia' visto la radice quadrata:

```
10 LET A=SQR(25)
```

- 25 e' il valore in ingresso;

- SQR( ) e' la scatoletta

- 5 (il valore della funzione calcolata) e' il dato in uscita.

Le funzioni sono molto comode e consentono di valutare semplicemente delle grandezze che in caso diverso avrebbero richiesto calcoli complicati. Ad esempio le funzioni trigonometriche SIN, COS, TAN.

E' importante comunque osservare che le funzioni matematiche possono essere sempre "surrogate" con l'uso delle quattro operazioni o di altre funzioni e questo spiega perche' il BASIC ne sia relativamente povero al confronto di certe calcolatrici scientifiche che non possiedono pero' la stessa flessibilita' nel costruirle.

Anche senza spingersi nei dettagli della trigonometria, vediamo come e' facile disegnare la funzione COS(X) sul nostro monitor:

```
10 GRAPH
20 ORG 0,95
30 PLOT 0,0,255,0
40 FOR I=0 TO 255
```

```
50 SET I,COS(I*6.28/255)*90
60 NEXT
70 AT 22,5
80 PRINT "GRAFICO DI COS(X)"
```

Le funzioni trigonometriche hanno moltissime applicazioni; tra di esse va segnalata, se non per importanza ma certo per popolarita', quella del tracciamento dei diagrammi dei bioritmi.

Nel capitolo XIII e' riportato un dettagliato elenco di tutte le funzioni disponibili nel GBASIC.

Valutazione di una funzione

Una funzione in una espressione aritmetica si comporta come un numero, o, per meglio dire, viene, come una variabile, ridotta al numero che ne rappresenta il valore. Il BASIC calcola cioe' prima quello che sta tra le parentesi della funzione (l'argomento) se questo non e' scritto direttamente sotto forma di un numero (ad esempio SQR(25+5)) e poi calcola la funzione. A questo punto l'intera funzione viene sostituita dal valore calcolato.

Esempio:

```
10 A=9
20 LET B=3+SQR(40+A)
```

Prima si calcola l'argomento della funzione (40+9=49).

Poi si esegue la radice (radice di 49=7).

Quindi si sostituisce tutta la funzione con il suo valore (7).

Infine si valuta la parte rimanente della espressione in cui la funzione e' inserita.

CAPITOLO XII
La stampante

La stampante

La stampante e', come dice il nome, una macchina che, collegata ad un elaboratore elettronico consente a quest'ultimo di emettere informazioni stampate su un pezzo di carta.

Esistono moltissimi tipi di stampanti, differenti per la tecnica di scrittura e per le prestazioni ma tutte concettualmente identiche. Non descriveremo qui pertanto la stampante come dispositivo fisico ma come <u>unita'</u> <u>logica</u>.

Possiamo immaginare la nostra unita' logica come una macchina per scrivere collegata all'elaboratore; il modo in cui il collegamento avviene non e' molto importante, potendo variare da caso a caso.

Il comportamento della stampante e' molto semplice; sul nostro computer disponiamo di agevoli comandi per il suo azionamento. In LPRINT LLIST particolare abbiamo i comandi LPRINT e LLIST, del tutto identici ai comandi PRINT e LIST ma con uscita sulla stampante invece che sul video.

Abbiamo gia' visto piu' volte l'uso della frase PRINT. Ad esempio:

```
10 PRINT "Il risultato e' ",10/2
```

provoca la comarsa sul video della scritta

```
Il risultato e' 5.
```

Alla stessa maniera, la frase

```
10 LPRINT "Il risultato e' ",10/5
```

provoca la scrittura sul foglio della stampante del messaggio

Il risultato e' 5.

L'uso di PRINT ed LPRINT consente quindi di sterzare a piacimento le uscite tra video e stampante senza che tra i medesimi vi sia alcuna interazione; le uscite sono del tutto indipenenti.

Similmente avviene per il comando LLIST che provoca la stampa anziche' la visualizzazione del listato del programma.

L'impiego della stampante allarga le possibilita' di qualsiasi computer, sia perche' gli fornisce una via per emettere risultati scritti sia perche' consentendo la stampa del listato di un programma facilita molto l'opera del programmatore che non e' piu' costretto a ricostruire le sue procedure attravrso il video.

CAPITOLO XIII
Istruzioni, funzioni e comandi

Come usare questa sezione

In questa sezione sono descritte in ordine alfabetico, tutte le frasi, le funzioni ed i comandi che compongono il GBASIC.

Questa parte e' piu' da consultare piu' che da leggere. Cionnonostante una rapida scorsa potra' essere utile al lettore sia come ripasso che per conoscere tutti i dettagli non ancora forniti nei capitoli precedenti.

Aritmetica

Il GBASIC tratta numeri in virgola mobile con 6 cifre significative e campo dinamico compreso tra 10 alla 127 e 10 alla -128. Questo vuol dire che si possono trattare numeri grandissimi con 127 zeri o numeri piccolissimi preceduti dalla virgola e da 127 zeri.

Numeri di linea

Nel GBASIC i numeri delle linee di programma devono essere compresi tra 1 e 65535.

Variabili

Le variabili ammesse sono 286 di tipo scalare (cioe' senza indici) piu' 286 di tipo vettoriale, con un indice compreso tra 0 e 255. La memoria disponibile limita ovviamente il numero complessivo delle variabili utilizzate ma non la loro scelta. I nomi delle variabili sono costitui-

ti da una lettere dell'alfabeto inglese o da una lettera dell'alfabeto inglese seguita da una cifra.

Esempi:

A
A1
A2
..ecc

Le variabili ad indice vengono dimensionate automaticamente salvo diversa indicazione (vedi frase DIM) al valore 10. Poiche' e' presente anche l'elemento zero, il dimensionamento automatico da' luogo a 11 variabili.

L'utilizzazione dello spazio e' ottimizzata, nel senso che e' possibile utilizzare, a parita' di memoria, grandi programmi e poche variabili o piccoli programmi e tante variabili. L'utente non deve procedere a stabilire le assegnazioni degli spazi che sono automatiche.

Alfabeto

Il GBASIC utilizza i seguenti caratteri:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
01234567890
+ - * / = < >
( ) " % , ; .

i caratteri sotto riportati possono invece essere utilizzati ma solo racchiusi, assieme ai precedenti o da soli, tra doppi apici nelle frasi PRINT, LPRINT ed INPUT:

abcdefghijklmnopqrstuvwxyz
! # $ & ' : { } [ ] ^ ~ @ ` | \ ?

I programmi quindi, con la sola eccezione delle frasi esplicative di PRINT, LPRINT ed INPUT, devono essere scritti in lettere maiuscole o risultera' un ERRORE DI LINGUAGGIO.

Editing

La semplice battitura di linee precedute dal relativo numero provoca automaticamente l'ordinamento delle stesse in senso crescente. E' bene prevedere numeri debitamente intervallati per facilitare successive inserzioni.

Una linea viene cancellata semplicemente battendo il suo numero seguito subito da RETURN.

Una linea viene corretta riscrivendola.

Eventuali errori di battitura possono essere immediatamente corretti tornando indietro con il tasto freccia a sinistra.

Una intera linea viene cancellata durante la battitura con il tasto CTRL/C, utile anche per interrompere un programma in corso.

Esecuzione differita

Quasi tutte le frasi ed i comandi sotto elencati possono essere usati per l'esecuzione immediata o differita. Quelli che devono necessariamente essere usati in modo immediato o differito sono indicati rispettivamente con (imm.) o con (diff.).

Per esperti

Le frasi o le funzioni che richiedono conoscenze avanzate di programmazioni in linguaggio macchina sono indicate con E+. La loro spiegazione e' sintetica in quanto si rivolge a persone che gia' dispongono di adeguata formazione. Si suggerisce ai principianti una attenta lettura degli articoli gia' comparsi o che compariranno su **cq elettronica.**

Abbreviazioni

La corretta definizione di un linguaggio viene di solito effettuata in forma abbastanza complessa e di difficile interpretazione per il principiante. Attenendoci ai principi ispiratori di questo manuale, anche qui faremo eccezione e ci baseremo su una simbologia meno rigorosa ma intui-

tiva. Tutte le abbreviazioni sotto riportate richiamano concetti gia' ampiamente illustrati nelle sezioni precedenti.

esp - sta per espressione aritmetica
var - sta per variabile
linnum - sta per numero di linea

Altre abbreviazioni sono spiegate nel testo.

Quando una frase prevede una serie di parametri di numero non definito, si usano i puntini di sospensione; ad esempio

INPUT var,var1,var2...

indica che dopo INPUT possono essere indicate una serie di variabili, var, var1 ecc.

Per la PRINT, che avrebbe richiesto una descrizione della forma generale troppo complessa, si e' ricorsi alla definizione a mezzo di una ampia serie di esempi.

F R A S I

AT

Forma generale:

AT espl,esp2

Descrizione: posiziona la prossima uscita su video alla riga espl ed alla colonna esp2. Viene usata per dirigere una uscita in un punto assegnato dello schermo. La prima riga e' 0 ed il primo carattere e' 0.

Limiti: in modo TEXT espl deve dare un risultato compreso tra 0 e 23 ed esp2 deve dare un risultato compreso tra 0 e 39. In modo GRAPH invece espl deve stare sempre tra 0 e 23 ma esp2 tra 0 e 31.

Esempi:

```
10 AT 20,1020
20 PRINT "SONO ALLA RIGA 20 COLONNA 10"

10 LET A=10
20 LET B=0
30 AT A,B
40 PRINT 123/5

10 AT 1,8
20 LET A(9)=10
30 LET S=56/T-A(9)
40 LPRINT S
50 PRINT "SONO A RIGA 1 COLONNA 8"
```

BLANK

Forma generale:

BLANK

Descrizione: La frase BLANK serve per disattivare il video del G5. La memoria e le funzioni video restano attive ed inalterate. Serve ad esempio per preparare una pagina video senza farla vedere all'operatore o per giochi. L'effetto della frase BLANK viene annullato dalla frase NOBLANK che fa

ricomarire l'immagine sullo schermo. L'effetto di BLANK/NOBLANK e' esattamente quello che si avrebbe rispettivamente abbassando o sollevando una tendina opaca davanti al video.

Esempio: (la subroutine 1000 disegna sul video l'immagine di un cane)

```
10 PRINT "Adesso ti mostrero' per un attimo"
20 PRINT "un immagine. Dovrai dire cosa"
30 PRINT "rappresenta."
40 FOR I=1 TO 100
50 NEXT
55 GRAPH
60 BLANK
70 GOSUB 1000
80 NOBLANK
90 REM BREVE RITARDO
100 FOR K=1 TO 250
110 NEXT
120 REM PULIZIA VIDEO
130 TEXT
140 PRINT "Gatto = 1"
150 PRINT "Elefante = 2"
160 PRINT "Serpente =3"
170 PRINT "Cane =4"
180 PRINT
190 INPUT "Cos'era? ";I
200 IF I=4 THEN PRINT "BRAVO!"
210 IF I<>4 THEN PRINT "SBAGLIATO"
220 END
```

CLEAR

Forma generale:

CLEAR

Descrizione: CLEAR azzera tutte le variabili ponendo il loro valore a zero.

Esempio:

```
10 CLEAR
20 PRINT A

RUN

 0
```

DATA
(diff.)

Forma generale:

DATA val1,val2,val3...

Descrizione: La frase data crea un elenco di valori che possono successivamente essere letti dalla frase READ che li assegna a delle variabili. Possono essere usate piu' DATA; l'elenco dei valori viene costituito dall'insieme di tutte le frasi DATA, essendo il primo elemento dell'insieme il primo valore della prima frase DATA e l'ultimo l'ultimo della ultima frase DATA. In un programma la frase READ non puo' precedere la frase DATA ma peraltro le frasi DATA possono essere poste ovunque. I valori val1, val2 ecc. devono essere scritti esatamente come si introdurrebbero in risposta ad una frase INPUT (vedi).

Esempi:

```
10 DATA 12.6,7,897
20 READ A,B,C
30 PRINT A,B,C

RUN

 12   6.7  897

10 DATA 3,6
20 DATA 6,8
30 READ A,C,C1,C2
40 PRINT A,C,C1,C2

RUN

 3  6  6  8
```

DIM
(diff.)

Forma generale:

DIM var(num),var1(num1)....

Descrizione: La frase DIM serve per riservare in memoria lo spazio per una o piu' variabili ad in-

dice di nome var, var1 ecc. La variabile var avra' dimensione num, la variabile var1 avra' dimensione num1 e cosi' via. Se num vale 12, la variabile var avra' 13 elementi, da var(0) a var(12).
Variabili con indici compresi tra 0 e 10, quindi con meno di 11 elementi, vengono dimensionate automaticamente e non e' quindi richiesto l'uso della frase DIM.

Esempi:

```
10 REM NON RICHIEDE DIM
20 LET A(8)=12

10 DIM A(30),B(20)
20 LET A(12)=0
30 LET A(A(12))=8
```

Si noti che la frase DIM usata in modo diretto segnala ERRORE DI DIMENSION e non di MODO DIRETTO.

END

Forma generale:

END

Descrizione: La frase END deve sempre essere posta al termine di ogni programma ed indica la fine fisica del medesimo.

Limiti: Quando si omette la frase END non compaiono segnalazioni di errore ma possono derivare inconvenienti da detta mancanza usando le variabili ad indice.

Esempio:

```
.....
1023 PRINT "fine del programma"
1030 END
```

FOR
(diff.)

Forma generale:

FOR var=esp1 TO esp2 STEP esp3

Descrizione: La frase FOR viene usata per eseguire ciclicamente tutte le istruzioni che stanno fra la

frase FOR medesima e la prossima frase NEXT. La frase FOR dice: "Esegui tutte le istruzioni che seguono fino alla NEXT per la variabile var che va dal valore esp1 al valore esp2, incrementandola ogni volta del valore esp3". Se STEP esp3 viene omesso, alla variabile var viene aggiunto ad ogni ciclo il valore 1. Tutte le espressioni possono assumere qualsiasi valore.

Limiti: la frase NEXT deve essere presente o risultera' un ERRORE DI FOR..NEXT.

Esempi:

```
10 FOR I=1 TO 24
20 PRINT TAB(I),"*"
30 NEXT
```

Spiegazione: la frase 20 viene eseguita per I che va da 1 a 24. Poiche' non e' specificato STEP, ad ogni ciclo viene sommato 1 alla variabile I. I vale quindi 1 al primo giro, 2 al secondo e cosi' via.

```
10 FOR H1=20 TO 18.5 STEP -0.5
20 PRINT SQR(H1)
30 PRINT H1*H1
40 NEXT
```

Spiegazione: le frasi 20 e 30 vengono eseguite per H1 che va da 20 a 18.5; all'indietro perche' STEP e' negativo. H1 vale 20 al primo giro, 19.5 al secondo e cosi' via.

GOSUB
(diff.)

Forma generale:

GOSUB linnum

Descrizione: Quando durante l'esecuzione viene incontrata la frase GOSUB viene eseguito un salto alla linea indicata da linnum. Non appena poi viene incontrata una istruzione RETURN si ritorna alla linea che segue la GOSUB.

Limiti: linnum deve essere un numero di linea

esistente o risultera' un ERRORE DI NUMERO DI LINEA.

Esempio:

```
10 GOSUB 1000
20 PRINT H1
30 STOP
1000 LET H1=45+7
1010 RETURN
```

Spiegazione: dalla frase 10 si salta alla 1000. Li' si esegue la LET.. ed alla frase RETURN si torna alla 20.

GOTO

Forma generale:

GOTO linnum

Descrizione: la frase GOTO serve per saltare al numero di linea linnum.

Limiti: linnum deve essere il numero di una linea esistente o si avra' una segnalazione ERRORE DI NUMERO DI LINEA.

Esempio:

```
10 GOTO 300
....
300 PRINT A
```

GRAPH

Forma generale:

GRAPH

Descrizione: la frase GRAPH e' utilizzata per predisporre il modo grafico. Alla sua esecuzione il video si pulisce ed il formato di visualizzazione alfanumerica diviene di 24 righe di 32 caratteri. Il cursore non compare piu'. Quando si raggiunge l'ultima linea non si ha lo scorrere automatico della pagina ma la scrittura prosegue sull'ultima riga sovrapponendosi. Il comando TEXT

(vedi) e' usato per tornare nel modo alfanumerico TEXT.

Limiti: nessun comando relativo alla grafica o agli sprites puo' essere usato senza essere prima prima passati in modo grafico con il comando GRAPH.

Esempi:

```
10 GRAPH
20 PLOT 12,12,3,4
30 AT 10,16
40 PRINT "RAGGIO =";R
```

```
10 GRAPH
20 SPMOV 1,0,80
```

IF..THEN

Forma generale:

IF esplog THEN frase1

Descrizione: la frase IF viene usata per eseguire frase1 se si verificano certe condizioni, se cioe' l'espressione logica esplog e' vera. Se cio' non avviene l'esecuzione continua dalla linea subito seguente e la parte che segue THEN viene ignorata. L'espressione esplog e' cosi' costruita:

esp1 oprel esp2

dove esp1 ed esp2 sono due comuni espressioni aritmetiche e oprel e' un operatore di relazione scelto tra quelli che seguono:

= uguale
> maggiore
< minore
<= minore od uguale
>= maggiore od uguale.
<> diverso

Esempi:

```
10 IF A=0 THEN PRINT "ERRORE"
20 ..
```

Spiegazione: se A vale 0 allora scrivi errore e passa alla 20. Se A e' diverso da 0 passa subito alla 20.

```
10 IF W1+W2<E+K THEN LET A=89
20 ..
```

Spiegazione: se la somma di W1+W2 e' minore della somma di E+K poni A=89 e passa alla 20. Se non passa direttamente alla 20.

```
10 IF 1=2 THEN PRINT "FOLLIA"
20 ..
```

Spiegazione: poiche' 1 non sara' mai uguale a 2, la frase PRINT "FOLLIA" non sara' mai eseguita!

INPUT

Forma generale:

```
INPUT "stringa",var,var1..
```

Descrizione: La frase INPUT e' utilizzata per richiedere all'operatore di introdurre il valore di una o piu' variabili var, var1 ecc. Opzionalmente puo' essere inserita una stringa di caratteri racchiusa tra doppi apici che ricordi e descriva brevemente il significato delle quantita' richieste. Quando la frase INPUT viene incontrata l'elaborazione si arresta e compare un punto interrogativo (se non e' stata utilizzata la stringa). L'operatore puo' cosi' introdurre i dati richiesti; se essi sono piu' di uno li separera' con una virgola ed infine premera' RETURN per far ripartire il programma.

Limiti: se si batte subito RETURN senza introdurre i dati richesti si genera un ERRORE DI INPUT. Se si batte RETURN prima di avere introdotto tutti i dati compaiono due punti interrogativi ed e' possibile continuare l'introduzione.

Esempi:

```
10 INPUT A
```

```
10 INPUT "Base ed altezza? ",B,A
```

```
10 INPUT "Costo unitario = ",C1
```

Nota: la frase INPUT usata in modo diretto provoca la segnalazione di un ERRORE DI LINGUAGGIO e non di MODO DIRETTO.

LET

Forma generale:

```
LET var=esp
```

Descrizione: la frase LET serve per assegnare ad una variabile un certo valore risultante da una espressione aritmetica.

Esempi:

```
10 LET A=7.9

10 LET C = 3 / 8

10 LET C=3-9

10 LET D1=(X-Y)/(X+Y)

10 LET F=SIN(3.21)/COS(A)

10 LET R=1000
```

LPRINT

Forma generale:

```
LPRINT "stringa",var,var1..
LPRINT "stringa",var,"stringa1",var1..
```

Descrizione: la frase LPRINT e' identica alla frase PRINT ma dirige l'uscita dell'elaboratore sulla stampante anziche' sul video. Vedi la frase PRINT.

Limiti: la frase LPRINT non dispone delle stesse funzioni CHR() come la PRINT; la specifica CHR puo' essere ugualmente usata ma il suo effetto e' legato al tipo di stampante impiegata. Vedere il manuale della stampante stessa.

Esempi:

```
10 LPRINT "Raggio ",A,B,C

20 LPRINT "Lunghezza ",L," metri"
```

NEXT
(diff.)

Forma generale:

NEXT

Descrizione: indica la fine di un ciclo di FOR. Piu' cicli di FOR possono essere innestati uno dentro l'altro; per ogni FOR e' necessaria una NEXT. L'accoppiamento avviene in modo automatico tra l'ultimo FOR e la prima NEXT (vedi anche frase FOR).

Limiti: se viene incontrata una frase NEXT senza che questa sia stata preceduta da un FOR si ottiene un ERRORE DI FOR..NEXT. Similmente nel caso in cui una frase NEXT risulti mancante.

Esempi:

```
10 FOR I=1 TO 10
20 PRINT
30 NEXT

10 FOR K=1 TO 10
20 FOR J=1 TO 8
30 LET A(J)=J
40 NEXT
50 LPRINT D(I)
60 NEXT
```

NOBLANK

Forma generale:

NOBLANK

Descrizione: l'effetto della frase NOBLANK e' quello di riattivare l'uscita video sospesa con la frase BLANK (vedi).

Esempio:

```
100 NOBLANK
```

ORG

Forma generale:

ORG esp1,esp2

Descrizione: ORG e' utilizzata per spostare in un punto arbitrario l'origine degli assi cartesiani di riferimento per le istruzioni relative alla grafica. Normalmente l'origine si trova nell'angolo in basso a sinistra (coordinate 0,0). In alto a destra e' invece presente il punto 255,191. Esp1 ed esp2 danno, con tali riferimenti, la nuova posizione del punto di coordinate 0,0.

Limiti: esp1 ed esp 2 devono essere compresi tra 0 e 255.

Esempi:

```
10 GRAPH
20 ORG 127,95
```

Spiegazione: il punto di coordinate 0,0 viene posto al centro dello schermo. Le coordinate x negative vanno verso sinistra e le y negative verso il basso del video.

```
30 ORG 0,0
```

Spiegazione: l'origine viene riposta nell'angolo in basso a sinistra.

OUT
E+

Forma generale:

OUT esp1,esp2

Descrizione: OUT serve per eseguire l'emissione di singoli valori su una porta di uscita. Esp1 e' il valore da far uscire ed esp2 e' il numero della porta. Gli indirizzi delle porte dipendono dall'indirizzo base della scheda usata come riportato nella tabella seguente (tra parentesi gli indirizzi in codice esadecimale).

Connettore	Indirizzo base	
J1	124	(7C)
J2	188	(BC)
J3	220	(DC)
J4	236	(EC)

su scheda madre:

tastiera	dati	244	(F4)
"	ctr.	246	(F6)
PIO	dati	245	(F5)
"	ctr.	247	(F7)
Video	ram	250	(FA)
"	reg.	251	(FB)

Ai valori base sopra riportati deve essere sommato il numero della porta sulla scheda (0, 1, 2 o 3). Vedere per maggiori dettagli gli articoli su **cq elettronica.**

Limiti: Sia esp1 che esp2 devono essere compresi tra 0 e 255 o ne risultera' un ERRORE DI LIMITE.

Esempi:

```
10 OUT 128,124
```

PLOT

Forma generale:

```
PLOT esp1,esp2,esp3,esp4
```

Descrizione: in modo grafico con la frase PLOT vengono tracciati segmenti di retta tra due punti, il primo di coordinate esp1,esp2 ed il secondo esp3,esp4. Se il tracciamento avviene sopra caratteri gia' scritti, questi non vengono alterati. I tracciati ottenuti con la frase PLOT si trovano sul piano prospettico piu' lontano dall'osservatore; essi sono quindi nascosti da un eventuale sprite.

Limiti: le coordinate dei punti devono essere tali da màntenere i tracciamenti entro l'area di 255x255 in cui e' ammessa la grafica e di cui e' mostrata su video la porzione inferiore di 255x191; in particolare dovra' essere posta atten-

zione quando l'origine e' spostata dall'angolo in basso a sinistra.

Esempi:

```
10 PLOT 0,0, 23,90

10 PLOT X,Y,X1,Y1

10 PLOT X,Y,X+10,Y+10
```

POKE
E+

Forma generale:

POKE esp[esp1,esp2..]

Descrizione: la frase POKE e' utilizzata per scrivere dei contenuti in una cella di memoria ad un certo indirizzo assoluto e nei successivi. Esp da' l'indirizzo ed esp1, esp2 ecc. danno i relativi contenuti; esp1 va all'indirizzo esp; esp2 a esp+1 e cosi' via.

Limiti: esp deve essere compreso tra 16384 e 65535 (infatti la memoria sotto 16384 e' di tipo in sola lettura e non vi possono quindi essere inseriti dei valori). La locazione di memoria piu' alta effettivamente disponibile dipende dalla schede di memoria inserite. Esp1, esp2 ecc. devono essere comprese tra 0 e 255.

Esempi:

```
10 POKE 20000[0]

10 POKE 20000[0,9,89,234,90,76]
```

PRINT

Forma generale:

PRINT "stringa1",esp1
(molte varianti ammesse, vedi esempi)

Descrizione ed esempi: la frase PRINT serve per determinare l'uscita sul video di valori numerici e di eventuali scritte esplicative.

Esempio:

```
5 A6=3.678
10 PRINT "Il risultato e' ",A6

RUN

Il risultato e' 3.678
```

La stessa frase PRINT puo' essere impiegata per stampare piu' dati e piu' frasi.

Esempio:

```
10 PRINT A,A/4,A-8
20 PRINT "Tu pesi ",K1," chilogrammi"
```

PRINT TAB(X)

La frase PRINT puo' utilizzare la funzione TAB() per andare a scrivere ad una certa colonna.

Esempio:

```
10 PRINT TAB(30),"VALORE= ",V1
```

(per scrivere VALORE= alla colonna 30)

La funzione TAB() e' molto utile per generare diagrammi.

Esempio:

```
10 FOR I=1 TO 24
20 PRINT TAB(I),"*"
30 NEXT
```

FORMATI

La frase PRINT consente anche di variare il formato dei dati in uscita. Per fare cio' si usano dei codici preceduti e seguiti dal segno %.

Esempi:

```
10 PRINT %2%
```

(dispone la stampa di 2 posti decimali dopo il punto con arrotondamento automatico. Se i decimali sono nulli non vengono stampati).

```
10 PRINT %Z2%
```

(come la precedente. Le due cifre decimali vengono pero' stampate anche se sono zero)

10 PRINT %Z%

(pone sempre tutti i decimali possibili)

10 PRINT %E%

(passa alla notazione esponenziale indicando tutti i decimali che servono)

10 PRINT %E2%

(passa alla notazione esponenziale indicando fino a 2 decimali se diversi da zero)

10 PRINT %ZE2%

(passa alla notazione esponenziale mettendo sempre due decimali anche se nulli)

FUNZIONI

La frase PRINT serve poi, in modo TEXT, per attivare certe particolari funzioni del video con la funzione CHR() che consente di inviare caratteri speciali di controllo.

Esempio:

Pulizia video

10 PRINT CHR(12)

(provoca la pulizia del video)

Sottolineatura

20 PRINT CHR(22),"MARAMAO",CHR(23)

(provoca la scrittura della parola MARAMAO con sottolineatura).

Queste le funzioni disponibili:

Codice	Funzione
8	Passo indietro cursore
9	Tabulazione (nota 1)
10	Interlinea
11	Cursore in alto a sinistra
12	Pulizia video
13	Ritorno a capo
14	Passo avanti cursore
15	Passo in alto cursore

22	Inizio sottolineatura
23	Fine sottolineatura

Nota 1 - Il codice 9 provoca l'avanzamento del cursore fino al prossimo arresto di tabulazione. Gli arresti sono alle colonne 7, 15 e cosi' via di 8 in 8 colonne.

ATTENZIONE PERO'! In modo GRAPH sono attive solo le funzioni 10 e 13.

POSIZIONAMENTO

Il posizionamento delle uscite determinate dalla PRINT viene eseguito con la frase AT (vedi).

PUSH
E+

Forma generale:

PUSH esp1,esp2...

Descrizione: la frase PUSH e' stata creata per facilitare il collegamento con routines in linguaggio macchina. I valori esp1, esp2 ecc. vengono "push-ati" nello stack e possono essere poi "pop-ati" dalla routine dell'utente.

Limiti: i valori esp1, esp2 ecc. devono essere compresi tra 0 e 65535.

Esempio:

```
10 PUSH 23,7876,567
```

READ
(diff.)

Forma generale:

READ var1,var2...

Descrizione: la frase READ assegna alla variabile var1 il primo valore letto dall'insieme delle frasi DATA (vedi), alla seconda il secondo e cosi' via. Alla fine il puntatore di lettura viene lasciato posizionato dopo l'ultimo elemento letto. La successiva frase READ partira' quindi da esso per le successive assegnazioni. La frase READ e' utile per eseguire lunghi caricamenti di variabili. Desiderando riposizionare all'inizio dell'in-

sieme DATA il puntatore, si usera' la frase RESTORE (vedi).

Limiti: se si tenta di leggere variabili quando non ci sono piu' DATA disponibili si ottiene un ERRORE DI READ.

Esempio:

```
10 DATA 12.4,45,8
20 READ A,B,C
```

REM

Forma generale:

REM testo

Descrizione: la frase REM serve per aggiungere commenti all'interno di un programma. Il testo viene completamente ignorato.

Limiti: le frasi REM accrescono il tempo di esecuzione di un programma e aumentano la sua occupazione di memoria.

Esempi:

```
10 REM TRALLALLA' TRALLALLA'
```

```
10 REM ROUTINE DI CALCOLO POSIZIONE
```

```
10 REM PROGRAMMA LUNA
20 REM VERSIONE 1.0 DEL 12/7/82
```

RESET

Forma generale:

RESET esp1,esp2

Descrizione: RESET serve per spegnere, in modo GRAPH, un punto sullo schermo di coordinate esp1,esp2.

Limiti: le coordinate devono rientrare nel campo dello schermo. Se il punto indicato non e' illuminato la frase RESET non ha effetto.

Esempio:

```
10 RESET 10,100
```

RESTORE
(diff.)

Forma generale:

RESTORE

Descrizione: la frase RESTORE serve per riposizionare il puntatore della frase READ all'inizio dell'insieme DATA.

Esempio:

```
10 DATA 12,34,45,67
20 FOR I=1 TO 4
30 READ A(I)
40 NEXT
50 RESTORE
60 READ X,Y,Z,W
70 PRINT A(1),A(2),A(3),A(4),X,Y,Z,W

RUN

12  34  45  67  12  34  45  67
```

RETURN
(diff.)

Forma generale:

RETURN

Descrizione: serve per ritornare alla frase successiva all'ultimo GOSUB (vedi) eseguito.

Limiti: se nessuna frase GOSUB era stata precedentemente eseguita si ha un messaggio ERRORE DI FOR..NEXT.

Esempio:

```
10 GOSUB 1000
20 PRINT A
30 STOP
1000 A=787878
```

```
1010 RETURN
```

SET

Forma generale:

SET espl,esp2

Descrizione: la frase SET serve per illuminare, in modo GRAPH, un punto sullo schermo di assegnate coordinate espl, esp2. Il punto puo' poi essere spento con la frase RESET.

Limiti: le coordinate devono rientrare nel campo dello schermo. Se il punto e' gia' illuminato la SET non ha effetto.

Esempi:

```
10 SET 10,67
```

```
10 LET A=60
20 LET B=78
30 SET A,B
```

STOP
(diff.)

Forma generale:

STOP

Descrizione: quando in un programma viene incontrata la frase STOP l'esecuzione si arresta e compare il messaggio:

STOP ALLA LINEA n

dove n e' il numero della linea dove si trova lo STOP. Il controllo ritorna all'operatore.

Esempio:

```
450 STOP
```

SPCOL

Forma generale:

SPCOL espl,esp2

Descrizione: SPCOL e' usata per definire il colore di uno sprite o l'intensita' di grigio nel caso in cui si usi un monitor bianco e nero. Espl indica lo sprite desiderato ed esp2 il colore da assegnargli.

Limiti: espl deve essere compreso tra 0 e 31, esp2 tra 0 e 15.

Esempio:

```
10 SPCOL 2,7
```

SPDEF

Forma generale:

SPDEF espl,esp2,iiiiiiiF

Descrizione: SPDEF serve per definire la forma del rigo esp2 dello sprite espl. I vari "i" vengono sostituiti da una X per indicare un punto acceso o da una O (lettera, non zero) per indicare un punto spento. Se da un punto illuminato fino a fine rigo i punti sono tutti spenti si puo' evitare di mettere tutti gli O ma basta mettere subito la F.

Gli sprites di taglia grande vengono definiti come 4 sprites di taglia piccola. Ciascuno dei piccoli compone un quarto del grande. Gli sprite grandi hanno la stessa forma a quattro a quattro.

Ad esempio lo sprite 0 di taglia grande e' composto dagli sprites piccoli 0, 1, 2 e 3 che determinano rispettivamente i quarti in alto a sinistra, in basso a sinistra, in alto a destra ed in basso a destra. In conseguenza di cio' gli sprites grandi 0, 1, 2 e 3 hanno tutti la stessa forma pur conservando la loro indipendenza.

Limiti: espl deve essere compreso tra 0 e 31; esp2 tra 0 e 7. Le "i" non devono essere sostituite che con X od O e non devono essere piu' di 8.

Esempi:

```
10 SPDEF 0,0,XXXOXOXOF
20 SPDEF 0,1,XF
30 SPDEF 0,2,OOOOOOOXF
```

SPFORM

Forma generale:

SPFORM esp1,esp2

Descrizione: generalmente ogni sprite ha la propria forma. Con la SPFORM e' possibile assegnare ad uno sprite esp1 la forma di un altro sprite esp2 che deve essere anche esso definito.

Limiti: esp1 ed esp2 devono essere compresi tra 0 e 31.

Esempio:

```
10 SPFORM 1,5
```

SPMOV

Forma generale:

SPMOV esp1,esp2,esp3

Descrizione: la frase SPMOV serve per posizionare lo sprite esp1 nel punto di coordinate esp2,esp3. Le coordinate si riferiscono al punto in alto a sinistra ello sprite stesso.

Limiti: esp1 deve essere compreso tra 0 e 31 e lo sprite deve essere definito. Le coordinate devono appartenere a punti dello schermo.

Esempio:

```
10 SPMOV 1,30,60
```

SPTYP

Forma generale:

SPTYP esp1,esp2,esp3

Descrizione: SPTYP serve per definire il numero e gli attributi degli sprites. Espl indica l'ingrandimento, esp2 la taglia e esp3 il numero degli sprite che, a partire dallo sprite 0, desideriamo attivare. Ingrandimento e taglia possono valere 0 od 1 e definiscono il tipo dello sprite; si hanno cosi' i tipi:

0-0 - sprite con reticolo di 8x8 punti.

1-0 - sprite con reticolo di 16x16 punti. Il programmatore non puo' pero' definirli tutti a suo piacimento; si tratta di un semplice ingrandimento dello sprite di tipo 0-0 in scala doppia.

0-1 - sprite con reticolo di 16x16 punti tutti definibili.

1-1 - sprite con reticolo 32x32 ottenuto come ingrandimento in scala doppia dello sprite 0-1.

Gli sprites di taglia grande vengono definiti per mezzo di 4 sprites piccoli (vedi anche SPDEF).

Limiti: espl ed esp2 possono valere solo 0 od 1. Esp3 deve essere compresa tra 0 e 31. La definizione del tipo e' uguale per tutti gli sprite in uso.

Esempi:

Per definire 4 sprite di tipo 0-1

```
10 SPTYP 0,1,4
```

Per definire 1 sprite di tipo 1-0

```
10 SPTYP 1,0,1
```

TEXT

Forma generale:

TEXT

Descrizione: la frase TEXT serve per disporre il modo di funzionamento TEXT. In modo TEXT il video e' arrangiato in 24 righe di 40 caratteri cadauna,

il cursore e' attivato e si ha lo scorrimento automatico verso l'alto (scroll) a fine pagina. Le funzioni video (vedi PRINT) sono tutte attive. Per usare la grafica si deve passare in modo GRAPH con l'omonimo comando (vedi).

Esempio:

```
10 TEXT
```

UNPLOT

Forma generale:

UNPLOT esp1,esp2,esp3,esp4

Descrizione: la fraase UNPLOT serve per spegnere, in modo GRAPH, tutti i punti posti tra i vertici del segmento che unisce i punti di coordinate esp1,esp2 ed esp3,esp4.

Limiti: le coordinate devono corrispondere a punti sullo schermo.

Esempio:

```
10 REM SCRIVE
20 PLOT 0,0,100,100
30 REM E CANCELLA
40 UNPLOT 0,0,100,100
```

VRAM
E+

Forma generale:

VRAM esp1,esp2

Descrizione: serve per settare ad un certo valore esp2 una locazione esp1 della memoria video (nel G5 infatti la memoria video non sottrae spazio all'area indirizzabile dall'unita' centrale).

Limiti: esp1 deve essere compreso tra 0 e 65535. Esp2 tra 0 e 255.

Esempio:

```
10 VRAM 0,1
```

FUNZIONI

ABS(X)

Funzione:

ABS(esp)

Descrizione: ABS(esp) ritorna il valore assoluto della espressione esp, cioe' se esp>=0 da' esp, se esp<0 da' -esp.

Esempi:

PRINT ABS(9876.8)

9876.8

PRINT ABS(-9876.8)

9876.8

ARG(X)
E+

Funzione:

ARG(esp)

Descrizione: ARG(esp) serve per passare argomenti a o da una routine dell'utente in linguaggio macchina.

Esempio:

LET A=ARG(45)

CALL(X)
E+

Funzione:

CALL(esp)

Descrizione: Chiamata ad una subroutine dell'utente in linguaggio macchina. Esp da' l'indirizzo della routine.

Esempi:

```
10 PRINT CALL(25000)

10 LET A=CALL(21987)
```

COS(X)

Funzione:

COS(esp)

Descrizione: COS(esp) ritorna il coseno della espressione esp. Esp deve essere in radianti.

Esempi:

```
10 D=COS(3*Y)

10 D=COS(A/B)
```

INP(X)

Funzione:

INP(esp)

Descrizione: INP(esp) ritorna il valore del dato in ingresso alla porta esp. Gli indirizzi delle porte sono riportate a proposito della frase POKE (vedi). La porta di ingresso della tastiera e' 241.

Limiti: Esp deve corrispondere ad una porta fisica realmente implementata.

Esempio:

```
10 PRINT INP(241)
```

INT(X)

Funzione:

INT(esp)

Descrizione: INT(esp) ritorna la parte intera di esp, cioe' la parte a monte del punto decimale.

Esempi:

PRINT INT(1)

1

PRINT INT(1.2)

1

PRINT INT(0.2)

0

PEEK(X)
E+

Funzione:

PEEK(esp)

Descrizione: PEEK(esp) ritorna il contenuto della cella di memoria all'indirizzo esp.

Limiti: esp deve essere compreso tra 0 e 65535 e deve ovviamente corrispondere a memoria realmente installata.

Esempio:

10 IF PEEK(21234)=0 THEN A=89

POP(X)
E+

Funzione:

POP(esp)

Descrizione: POP(esp) serve per estrarre un valore dallo stack di collegamento con il linguaggio macchina.

PVRAM(X)
E+

Funzione:

PVRAM(esp)

Descrizione: serve per ritornare il valore contenuto nella cella di memoria esp della memoria video del G5. Vedi anche VRAM.

Limiti: esp deve essere compreso tra 0 e 16384.

Esempio:

```
10 LET A=VRAM(890)
```

RND(X)

Funzione:

RND(esp)

Descrizione: RND(esp) ritorna un valore casuale compreso tra 0 ed 1.

Esempi:

```
PRINT RND(0)

 .098723

PRINT RND(0)

 .075670
```

SGN(X)

Funzione:

SGN(esp)

Descrizione: SGN(esp) ritorna la funzione segno di esp, cioe' 0 se se esp=0, 1 se esp>0, -1 se esp<0.

Esempi:

```
PRINT SGN(12)

 1

PRINT SGN(-12)

 -1
```

```
PRINT SGN(0)

 0
```

SIN(X)

Funzione:

SIN(esp)

Descrizione: SIN(esp) ritorna il seno di esp. Esp deve essere espressa in radianti.

Esempio:

```
10 LET A=123/SIN(X)
```

SPCOIN(X)

Funzione:

SPCOIN(esp)

Descrizione: SPCOIN(esp) serve per sapere se due o piu' sprites sullo schermo si stanno toccando, se hanno cioe' uno o piu' punti in sovrapposizione. SPCOIN(esp) vale 0 se non ci sono contatti ed 1 se ci sono. Esp viene ignorata. I punti non illuminati degli sprites non generano una coincidenza.

Esempio:

```
10 IF SPCOIN(0)=1 THEN PRINT "SCONTRO!"
```

SQR(X)

Funzione:

SQR(esp)

Descrizione: SQR(esp) ritorna la radice quadrata di esp.

Limiti: esp deve essere maggiore od uguale a zero.

Esempio:

```
PRINT SQR(25)

 5
```

TAN(X)

Funzione:

TAN(esp)

Descrizione: TAN(esp) ritorna la tangente di esp. Esp deve essere espressa in radianti.

Esempio:

```
10 LET S=TAN(X)
```

C O M A N D I

LIST
(imm.)

Forma generale:

```
LIST
LIST linnum
```

Descrizione: LIST serve per ottenere la lista del programma attualmente in memoria sul video. La lista puo' iniziare da una linea differente dalla prima specificando linnum. La lista si arresta alla fine del programma od alla pressione del CTRL/C da parte dell'operatore.

Esempio:

```
LIST 1000

1000 PRINT "Valore finale = ";T2
1010 END
```

LLIST
(imm.)

Forma generale:

```
LLIST
LLIST linnum
```

Descrizione: come LIST ma con uscita su stampante.

RUN
(imm.)

Forma generale:

```
RUN
```

Descrizione: RUN serve per avviare l'esecuzione di un programma a partire dalla linea di numero piu' basso. Tutte le variabili sono azzerate. Se durante l'esecuzione di un programma viene premuto un tasto senza che cio' sia richiesto dal programma stesso, sul video compare la scritta ESECUZIONE.

Esempio:

RUN

(l'esecuzione inizia..)

ERASE
(imm.)

Forma generale:

ERASE

Descrizione: il comando ERASE elimina il programma presente in memoria.

Esempio:

```
LIST

10 FOR I=1 TO 100
20 PRINT I
30 NEXT
40 END

ERASE

LIST

(nessun effetto)
```

INDICE

GIANNI
BECATTINI
IL COMPUTER È FACILE
I LIBRI
DELL'ELETTRONICA
7
edizioni
CD